NPS ARCHIVE
1998.09
HENCKE, R.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN AGENT-BASED APPROACH TO ANALYZING INFORMATION AND COORDINATION IN COMBAT

by

Richard B. Hencke

September 1998

Thesis Co Advisors: Donald P. Gaver
Carl R. Jones

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1998	Master's Thesis

4. TITLE AND SUBTITLE
AN AGENT-BASED APPROACH TO ANALYZING INFORMATION AND COORDINATION IN COMBAT

5. FUNDING NUMBERS

6. AUTHOR(S)
Hencke, Richard B.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

The quality and quantity of information flows is a critical factor in the command and control of forces in battle. Many current simulations do not adequately show the interactive effects of information on the battlefield. Agent-based simulation is a promising technique that can provide insight into these effects.

The purpose of this thesis was to develop an agent-based simulation to analyze the relationship between information and command structure. SInBaD (Simulation of Information in Battlefield Decisions) was the agent-based simulation developed specifically for this thesis. Although SInBad is only an abstract model of combat, it is believed that this approach can provide much insight into the mechanisms that affect the effectiveness of information in battle.

Several combat scenarios were simulated using different control rules. These simulations suggest that there exist scenarios where information is essential to mission success and some cases where its role is less instrumental or even detrimental.

Other insights generated from this research suggest that agent-based simulation may help define metrics useful in aiding decision-makers during the planning and execution of a large and complex campaign.

14. SUBJECT TERMS
Agent based simulation, Complexity Theory, Complex Adaptive Systems

15. NUMBER OF PAGES
107

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500
(Rev. 2-89)

Standard Form 298

Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

AN AGENT-BASED APPROACH TO ANALYZING INFORMATION AND COORDINATION IN COMBAT

Richard B. Hencke
Lieutenant, United States Navy
B.S., California State Polytechnic University, Pomona, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

11.9/13
H4130745
c.1

NPS ARCHIVE
1998.09
HENCKE, R.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

The quality and quantity of information flows is a critical factor in the command and control of forces in battle. Many current simulations do not adequately show the interactive effects of information on the battlefield. Agent-based simulation is a promising technique that can provide insight into these effects.

The purpose of this thesis is to develop an agent-based simulation to analyze the relationship between information and command structure. SInBaD (Simulation of Information in Battlefield Decisions) is the agent-based simulation developed specifically for this thesis. Although SInBad is only an abstract model of combat, it is believed that this approach can provide much insight into the mechanisms that affect the effectiveness of information in battle.

Several combat scenarios are simulated using different control rules. These simulations suggest that there exist scenarios where information is essential to mission success and some cases where its role is less instrumental or even detrimental.

Other insights generated from this research suggest that agent-based simulation may help define metrics useful in aiding decision-makers during the planning and execution of a large and complex campaign.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. PURPOSE OF RESEARCH 1
B. THESIS SCOPE 1
C. THESIS ORGANIZATION 2

II. BACKGROUND RESEARCH 5

A. NEW CONCEPTS IN FORCE STRUCTURE 5
1. The Role of Information in Battle 5
2. Network-Centric Warfare 5
3. Operational Maneuver from the Sea 7
B. CURRENT SIMULATION METHODOLOGIES 8
C. LESSONS IN COMPLEXITY 9
D. AGENT-BASED MODELING 12
1. Boids 13
2. Tierra 13
3. A Surrogate Market 14
E. MILITARY APPLICATIONS OF AGENT-BASED MODELING 15

III. DETAILS OF SINBAD 17

A. OVERVIEW 17
B. THE GENERAL SCENARIO 18
C. SIMULATING COMBATANTS 18
1. Agent Parameters 18
2. Agent Logic 20
D. THE ROLE OF INFORMATION 23
E. SIMULATING A COMMAND AND CONTROL STRUCTURE 24

IV. SIMULATION ANALYSIS 27

A. SIMULATION PHILOSOPHY 27
1. Tuning the Simulation 27
2. Winning and Losing and Risk 28
3. Battles on the Edge 29
B. VALIDATION 30
C. ANALYZING THE EFFECTS OF INFORMATION 34
1. High Tech vs. Numerically Superior 34
2. High Tech vs. Numerically Superior – Conclusion 39
3. Application of Information Warfare 42
4. Application of Information Warfare – Conclusion 47
D. A PLANNING AND REAL-TIME ANALYSIS TOOL 48
1. Combat Strategy Development 48
2. Identifying Risk and Critical Linkages 49
3. Real-Time Analysis 50
4. Example Risk Analysis 51

V. FUTURE PROJECTS 55

A. USER INTERFACE 55
B. INCREASED EFFICIENCY AND FIDELITY 56

C. ADAPTABILITY 56
APPENDIX. INSTALLATION AND PROGRAM CODE 59
A. USER'S MANUAL 59
1. Requirements 59
2. Installation 59
3. Running SInBaD 60
B. PROGRAM CODE 62
1. Input File 62
2. Sinbad.tk script 64
3. SInBaD source code 65
4. Sample Batch Mode Output 91
LIST OF REFERENCES 93
INITIAL DISTRIBUTION LIST 95

FIGURES

FIGURE 1 AGENT LOGIC CHART 20
FIGURE 2 AGENT'S LOGIC FLOW CHART 21
FIGURE 3 ATTRITION SCENARIO - INITIAL PLACEMENT 30
FIGURE 4 ATTRITION SCENARIO - ACTIVE ENGAGEMENT 31
FIGURE 5 FORCE LOCATION ATTRITION SCENARIO 32
FIGURE 6 ATTRITION SCENARIO - AGENT LOSSES AVERAGED OVER 10 RUNS 33
FIGURE 7 FORCE LOCATION FOR HIGH TECH (BLUE) VS. NUMERICALLY SUPERIOR (RED) - CASE 1 36
FIGURE 8 CASUALTIES FOR HIGH TECH (BLUE) VERSES NUMERICALLY SUPERIOR (RED) CASE 1 36
FIGURE 9 RED HERDING BEHAVIOR CASE 2 - 150 37
FIGURE 10 FORCE LOCATION FOR HIGH TECH (BLUE) VS. NUMERICALLY SUPERIOR (RED) – CASE 2 (150) 38
FIGURE 12 FORCE LOCATION FOR HIGH TECH (BLUE) VS. NUMERICALLY SUPERIOR (RED) – CASE 2 (200) 40
FIGURE 13 BLUE'S PROGRESS WITH VARYING SENSOR RANGE 40
FIGURE 14 FORCE LOCATION FOR APPLICATION OF IW - CASE 1 43
FIGURE 15 CASUALTIES FOR APPLICATION OF IW CASE 1 44
FIGURE 16 IW SCENARIO WITH RELATIVELY AGGRESSIVE BLUE AGENTS 45
FIGURE 17 FORCE LOCATION FOR IW SCENARIO WITH AGGRESSIVE BLUE AGENTS 46
FIGURE 18 CASUALTIES FOR IW SCENARIO WITH AGGRESSIVE BLUE AGENTS 46
FIGURE 19 FORCE LOCATION SUMMARY OF IW SCENARIO 47
FIGURE 20 FORCE LOCATION ATTRITION SCENARIO 52
FIGURE 21 ATTRITION SCENARIO WITH VARIABLE BLUE FORCE 53

TABLES

TABLE 1 LAND COMBAT AS A COMPLEX ADAPTIVE SYSTEM 11
TABLE 2 ATTRITION SCENARIO 31
TABLE 3 HIGH TECH (BLUE) VS. NUMERICALLY SUPERIOR (RED) (CASE 1) 35

I. INTRODUCTION

A. PURPOSE OF RESEARCH

The increased emphasis on Information Warfare (IW) and Information Operations (IO) has produced innovative ideas in the application of information technology. There is a natural assumption that an increase in communication is good but the validity of that belief has yet to be tested. In the era of decreasing defense budgets, it is ironic that our best hope for validating the effects of these new technologies lies in the technology itself. Computer simulation can provide much insight into the effects of introducing enhanced communications to the battlefield.

Of primary concern in this research is the relationship between information capability and command structure. The structure of a military fighting force is inextricably linked to its information infrastructure. If dramatic changes in this infrastructure were to occur, the requirement for a corresponding change in military structure would seem self-evident. Unfortunately, most current combat simulations have difficulty analyzing information's effects on the battlefield. Differential calculus based simulations (i.e. Lanchester based models) using aggregated data are too coarsely grained to reveal the interactive effects of information, and information network modeling (i.e. COMNET 3) analyze the physical effects on an information system but not how those effects alter the outcome of conflict.

This thesis is the result of the development of the SInBaD (Simulation of Information on Battlefield Decisions) computer simulation. SInBaD is a concept exploration simulator created to analyze information and its effects on Command and Control in combat. SInBaD is based on the precepts of Complexity Theory; a new approach to looking at dynamical systems that develops complex behavior through the interaction of its sub-systems.

With the help of SInBaD, this thesis explores force-on-force battlefield scenarios in order to gain insight into how information affects the outcome of combat.

B. THESIS SCOPE

The focus of this thesis can be divided into two major parts:

1. The development of an agent-based simulation to model the effects of information technology and related command and control structures on battlefield engagements.
2. The modeling of those effects by manipulating information based parameters and altering the composition and initial placement of battlefield forces.

There is no intention to accurately predict the effect of any specific information application to any specific combat scenario. It is hoped that the results of this exercise will provide general insight into the benefits and pitfalls of becoming a fighting force heavily reliant on information technology.

C. THESIS ORGANIZATION

There are five chapters and one appendix that comprise this research.

- Chapter I – Introduction. This chapter provides a general introduction to the focus of this research. The depth of the analysis and the organization of the thesis are also covered.
- Chapter II – Background Research. This chapter explores current thinking related to the application of information technology on the battlefield. A short overview of current simulation techniques as well as a brief discussion on Complexity Theory and related concepts is covered. Simulations based on these new theories are also reviewed.
- Chapter III – Simulation Overview. This chapter establishes the theoretical foundation of the algorithms used in SInBaD. A detailed discussion of the methods used to model information-based conflict is included.
- Chapter IV – Simulation Analysis. A justification and analysis of selected cases comprises this section. Several combat scenarios are modeled to determine what effect, if any, the increase in communication might have. Valid ranges for simulation parameters are established and justified.
- Chapter V – Future Projects. This research is far from complete. This section ponders the possible directions this research can go. Also, SInBaD is a simulation in its early stages. A discussion of its current weaknesses and possible fixes are presented.

- Appendix – User's Manual. Complete instructions on the installation and operation of SInBaD is presented here. Source code for SInBaD executables and other associated files are listed here.

II. BACKGROUND RESEARCH

A. NEW CONCEPTS IN FORCE STRUCTURE

1. The Role of Information in Battle

Information's significance in combat cannot be overstated. Whether it is to report a battle's status to a commander or transmit his orders to the field, its quality, in terms of timeliness and accuracy, drives command decisions that strongly influence the battle outcome. The very structure of a military organization is constrained by its ability to communicate. That structure is now experiencing fundamental change driven primarily by an increased ability to communicate. This revolution in information affairs has significantly impacted the U.S. commercial sector, providing a substantial profit incentive to develop information technology. With the military unable to match commercial research and development, they have surrendered the lead in establishing the direction of this technology. Fortunately, many of today's military leaders have recognized the need for change and have developed military concepts to capitalize on the information products being generated in the commercial sector.

Several new initiatives are capitalizing on the increased role of information. Since these new concepts helped generate the scenarios presented in Chapter 4, it is worthwhile to briefly review them.

2. Network-Centric Warfare

Several initiatives under consideration view information technology as a way to increase the tempo of battle. Vice Admiral Cebrowski has developed a concept he calls Network-Centric Warfare. His vision is a departure from the stove-piped "platform-oriented" thinking, which bases decisions on the abilities of specific platforms. Instead, he envisions a force that is highly inter-connected through an information grid. Engagement decisions are platform independent and rely on the total weapons capability in-theater. [CEB97] Ring-of-Fire is an example of a platform independent scenario in which a central authority controls fires from several platforms to support forces on shore.

Cebrowski's concept borrows heavily from the work of Brian Arthur, an economist currently at the Santa Fe Institue. Arthur's theory of "Increasing Returns"

states that some products (those that require an extensive support base) achieve market dominance if they can obtain an early foothold in the market. Arthur calls this "lock-out" because the competitor is effectively locked-out of the market when supporting products and a large consumer base establish the preferred product. The VHS-Beta war is a classic example where customers chose the convenience of VHS early on. This gave video suppliers an incentive to manufacture more VHS tapes, which in turn encouraged more customers to buy VHS. A positive feedback loop is created that drives all the other competition out of the market. [WAL92:1:17-18]

Vice Admiral Cebrowski develops two complementary concepts to attain lock-out as described in his Proceedings article:

- Network-Centric Warfare allows forces to develop speed of command.
- Network-Centric Warfare enables forces to organize from the bottom up – or to self-synchronize – to meet the commander's intent.

Speed of command has three parts:

1. The force achieves information superiority, having dramatically better awareness or understanding of the battlespace rather than simply more raw data.
2. Forces acting with speed, precision, and reach, achieve the massing of effects versus the massing of forces.
3. The results that follow are the rapid foreclosure of enemy courses of action and the shock of closely coupled events.

Vice Admiral Cebrowski's main point is that the "speed of command" of a network-centric force is increased to the point where an enemy commander is "locked-out" of pre-planned options, leaving only surrender or retreat as his only choices.

Of more relevance to this thesis is his assertion of the self-synchronizing ability of the network-centric force. He also emphasizes the need for "bottom up" organization to rapidly adapt to a complex situation.

His vision of a highly integrated mix of weapons platforms relies heavily on the ability of an information network to deliver accurate and timely information to commanders and targeting assets.

3. Operational Maneuver from the Sea

This concept, also known as Ship-to-Objective Maneuver (STOM), is a significant departure from previous Marine Corps-Navy doctrine. [STO98] In the past, the focus of amphibious warfare was the establishment of a large beachhead and involved the shuttling of large amounts of supplies from naval vessels close to shore. Operational Maneuver from the Sea eliminates the need for a beachhead by rapidly deploying amphibious forces over a large littoral zone from ships positioned over-the-horizon.

This concept expands the Marine Corps' idea of maneuver warfare out to the open sea. Combined Arms Teams embarked on landing craft, air cushion (LCAC) and advanced amphibious assault vehicles (AAAV) deploy with all the support they need to attain their onshore mission objective.

The STOM concept's command and control mechanism strikes a delicate balance between the offshore and on-scene commanders. The offshore commander conducts initial vectoring of combined arms teams. Once forces pass a predetermined Line of Departure (LOD), the on-scene commander has authority to control forces based on his tactical picture. Again, similar to the network-centric force, speed of command and self-synchronization are critical to maintaining the high operational tempo needed for mission success.

New advancements in transport and weapons technologies will make mobilization from far offshore possible, but it is the advancement in information capability that will allow for the level of synchronization required to rapidly place forces and fires on shore over a large littoral zone.

Maintaining battlespace awareness under a high operational tempo places a large burden on the supporting information infrastructure. As forces come ashore, the tactical scenario changes. Confrontation with the enemy, seizing objectives, and overcoming known and unknown obstacles revises the operational picture. The STOM concept also recognizes the importance of deception in opening gaps in enemy defenses that must be exploited quickly. Coordination of fires from offshore is also a primary component of the STOM concept and requires mechanisms for rapid target selection and continual BDA updates. All this information will reside on an information back-plane that must also be resistant to information warfare waged by the enemy.

Network-Centric Warfare and operational maneuver from the sea initiatives raise questions regarding the ability for an information infrastructure to provide the needed information. This thesis will attempt to provide justification for the reliance on information and identify possible weaknesses inherent in the reliance on a network view of the battlespace.

B. CURRENT SIMULATION METHODOLOGIES

Extensive research has been done in the area of combat modeling over the past several decades. The fidelity of these models range from highly aggregated to finely detailed. Highly aggregated models are used for budgeting, long-range logistical planning and in studying theater level operations. Highly detailed models help analyze, among other things, particular weapons platforms. All the models reviewed for this study are problem oriented. In other words, they are designed to solve some problem. Whether it is to help make budgeting decisions, determine the effectiveness of a weapons platform, or to ensure that enough material is available to perform a mission, this type of modeling seeks to find the optimal solution for a given set of requirements and a given set of constraints. [MOR95]

In this function, conventional combat modeling can be very effective. Their weakness lie in the their inability to explore the dynamics of warfare. The dynamics of the conventional combat model are "built in" from the beginning. They are the assumptions and constraints that arise when the simulation is first conceived. Exploration of the causes of the dynamics is not possible when the simulator predetermines them. J. Casti calls these models "prescriptive models"; they provide the analyst with an explicit way to control the dynamics of the model, usually through the adjustment of a variable or set of variables. [CAS97:18]

This is a fundamental difference between conventional simulations and agent-based simulation. An agent in an agent-based model contributes to an overlying dynamic by acting on its very limited rule set in conjunction with all the other agents. This approach to simulation allows for the study of why certain combat scenarios yield a particular result.

Another difficulty in the conventional combat modeling approach is its difficulty in dealing with *soft factors*. Many combat theorists today recognize the significant role of fighting spirit, morale, but modeling such concepts has been considered too difficult.

[ZIM98] Combat modelers are under increasing pressure to incorporate soft factors into their models. Many low-fidelity models use a single parameter to account for these factors but the resulting output appears contrived and does little to further understanding of soft factor's effects.

Of greater concern, with respect to this thesis, is the methodology's assumption of the massing of large forces that are endowed with perfect information and act in a completely rational manner. As mentioned earlier, new fighting concepts being proposed are based on relatively small groups of forces spread out over a large land region that must operate on potentially limited information. It is clear that a new modeling methodology is needed.

C. LESSONS IN COMPLEXITY

Before we explore a new methodology it will be useful to discuss some topics on which this simulation is based.

The field known as Complexity Theory has received wide spread attention only within the last few decades. It is more an approach to viewing dynamic systems than it is a theory like that found in the physical sciences. Prior to its application, most systems were analyzed using a reductionist approach. Decomposing systems into smaller sub-systems, identifying and cataloging parts, and then decomposing yet again into smaller sub-systems.

Neuro-biologists using this technique in their quest to understand the principles of the human mind ran into a wall of understanding as they reached the level of the individual neuron. How could the firing of electrical impulses between neurons generate such complex behaviors? A fundamental point of Complexity Theory, and the one critical to our discussion, is that the behavior of many systems is not derived from the collection of its parts whose effects can be linearly summed, but is due to the interaction between those parts acting with local information. Researchers are now postulating that it is the complex interaction of several thousand neurons that produces consciousness. [WAL92:5:145]

Dynamical systems that behave in this manner tend to be more capable of coping in a fluid environment. Many systems, especially in biology, achieve this ability by reorganizing their structure to adapt to a changing environment. These systems are called Complex Adaptive Systems (CAS).

John Holland, a principle architect in the field of complexity, identifies seven attributes of CAS. He breaks them up into three mechanisms and four properties: [HOL95:23]

Mechanisms

1. Internal Models – CAS use internal mental models to anticipate appropriate actions to environmental inputs. These models are developed from prior experience. Actions that yield beneficial outcomes are positively reinforced in the internal model, while negative results of an action are negatively reinforced.
2. Tagging – CAS possess a mechanism that allows for the identification of sub-systems. This is important to allow for filtering and communication.
3. Building Blocks – CAS use this mechanism to identify situations that may be completely novel. If a CAS enters a new situation, he may be able to recognize some familiarity by identifying parts of the situation that are familiar.

Properties

4. Non-linearity – CAS dynamics cannot be understood by the summing of its parts.
5. Flows – CAS are made up of agents who pass resources or information between themselves. An effect of an interaction may impact several agents as the resource or information is passed through the system. This creates a multiplying effect that is an inherent source of non-linearity.
6. Diversity – CAS are also characterized by possessing a wide range of agents. This is true in biological systems, where Nature has determined that it is too difficult to develop one super-agent to consume all resources. Instead, there are many different agents, working in cooperation and competition, each carving out a niche in the ecosystem. Diversity allows for simpler agents to maximize all the available resources. [WAL92:3:119]
7. Aggregation – CAS are hierarchical in nature. Biological systems are good examples. They are systems made up of cells, which in turn form organs that are sub-systems of the larger organism.

Table 1 is from A. Illachinski's 1997 report that accompanied the completion of ISAAC, an agent-based simulation that is discussed later in this thesis. In this table, he relates qualities of land combat to that of complex adaptive systems.

Table 1 Land Combat as a Complex Adaptive System

General Property of Complex Systems	Description of Relevance to Land Combat
Nonlinear interaction	Combat forces composed of a large number of nonlinearly interacting parts; sources include feedback loops in C2 hierarchy, interpretation of (and adaptation to) enemy actions, decision-making process, and elements of chance.
Nonreductionist	The fighting ability of a combat force cannot be understood as a simple aggregate function of the fighting ability of individual combatants.
Emergent Behavior	The global patterns of behavior on the combat battlefield unfold, or emerge, out of nested sequences of local interaction rules and doctrine.
Hierarchical structure	Combat forces are typically organized in a command and control (fractal-like) hierarchy.
Decentralized control	There is no master oracle dictating the actions of each and every combatant. The course of a battle is ultimately dictated by local decisions made by each combatant.
Self-organization	Local action, which often appears chaotic, induces long-range order.
Non-equilibrium order	Military conflicts, by their nature, proceed far from equilibrium; understanding how combat unfolds is more important than knowing the end state.
Adaptation	In order to survive, combat forces must continually adapt to a changing environment, and continually look for better ways of adapting to the adaptation pattern of their enemy.
Collectivist dynamics	There is a continual feedback between the behavior of (low-level) combatants and the (high-level) command structure.

Many military thinkers, Vice Admiral Cebrowski among them, have recognized that combat appears to act like a CAS. [CEB97] Combat is characterized by a diverse group of agents that act on internal mental models to carry out their mission. Information and supplies flow through the system, and the effects of those flows are widely felt throughout the system.

If combat is a CAS, then can we take advantage of that fact to help us understand how the dynamics of combat are affected by changes in information and soft factors? The next section will discuss some ideas that have developed, both in the civilian and military arena, using this approach.

D. AGENT-BASED MODELING

What makes an agent-based model so different? Agent-based models borrow heavily on a biological analogy. Biological systems are hierarchical in nature. The human body is a complex system made up of cells that form structures known as organs which, in turn, make up the structure of the human body. Entities, at any level in the biological hierarchy, act on local information to accomplish their goals. Cells interact with other cells in their immediate area, for example.

The premise in agent-based modeling is that many complex human systems develop their complexity through this local interaction. The goal, in this type of simulation, is to develop simple models for the individual agents in the level of hierarchy of interest, and then allow them to freely interact using their internal models. Even a handful of agents interacting can become extremely time-consuming to track using paper and pencil. Agent-based modeling is still in its infancy primarily because the computing power needed to perform this type of simulation is only now becoming widely available.

John Casti's book "Would–be Worlds" calls these types of simulations "artificial worlds". [CAS97] They are worlds populated by actors that are designed to possess the characteristics needed to adequately simulate the real life actors. The human controlling the simulation is an omnipotent being, adjusting the parameters of his actors to see how those changes affect his actor's behavior. The following are brief summaries of notable simulations culled from Casti's book as well as the A. Illachinski paper.

1. Boids

Many researchers have been using agent-based simulation to analyze biological systems. [CAS97:4:131-182] These simulations fall under the umbrella term "artificial life", a phrase coined by Chris Langton, the father of the field. It was at Langton's first A-Life conference that Craig Reynolds presented his Boids. [ISA97] Boids epitomizes the agent-based concept of discrete agents operating on a simple internal model and acting on local information. Boids follow three simple rules:

1. Maintain a minimum distance from other objects (including other boids).
2. Match the velocity of nearby Boids.
3. Move toward the perceived center of nearby birds.

With these three rules, Reynolds demonstrated his Boids exhibit the emergent behavior of flocking similar to that of birds. The flocking ability of the Boids is robust enough to allow individual Boids to regroup even when a barrier is encountered.

Boids is one of the first simulations to demonstrate that a complex globally emergent behavior could be generated from the actions of individual agents using local information and simple rules.

2. Tierra

Many of the early pioneers in agent-based modeling were not computer scientists. They were biologists and physicists who saw the computer as a tool to help them understand their field of study. Tom Ray is no exception. A naturalist from the University of Delaware, he spent 14 years in the jungles of South America before he decided that he could learn more about how animal life evolved by using a computer. He realized that the evolutionary process that created life on Earth had a sample size of one. Since it is unlikely that interstellar travel will be available to make observation of other planets possible, he decided to create his own worlds on a computer. [CAS:4:162-163]

Casti states that January 4, 1990 is the day the first non-Carbon based life was created. This was the day Tom Ray's simulation, Tierra, produced digital organisms that could self-replicate. Tierran organisms treat CPU time and memory as resources to be competed for. Digital Darwinism is the mechanism of selection that determines which simulated organisms will survive and reproduce. The organisms evolve and develop

strategies to use more resources. Those organisms that can "hog" resources have a better chance to reproduce.

The current version of Tierra is running on several computers across the globe. It is accessing the power of interconnected computers to answer questions about how life evolved on this planet.

Tierra is a good example of how agent based simulation can model complex systems that evolve and adapt to new selection pressures without the explicit hand of the simulator.

3. A Surrogate Market

In 1987, Brian Arthur and John Holland embarked upon the task of creating a simulation that could capture some of the complexity of the stock market. They realized that the current economic models used explicit models of human behavior which could not accurately reflect the dynamic and sometimes chaotic nature that is known as securities trading. [CAS97:2:77-81]

Using an agent-based approach, they modeled individual traders by giving them a set of market prediction tools (forecast tools that are commonly used to assist securities traders) and a rule base to determine which predictors they would use. The agents were a heterogeneous mix with one group of agents preferring some indicators that other groups did not.

Once the simulation was set in motion, no other input was added. The agents acting on their prediction tools bought and sold stocks based on those predictions. The resulting market behavior resembled that of the New York stock exchange. Instead of smooth curves representing the equilibrium dynamics of traditional economic theory, Arthur and Holland witnessed wild swings in the price of their stocks as the artificial market traders made buy and sell decisions.

The Surrogate Market is a classic example of a simulation that was not designed to find an answer to a specific question, like where is the stock market going next? It was an artificial world built to help understand the underlying dynamics that is securities trading.

Arthur and Holland's surrogate market demonstrated that agent-based modeling can provide insight into human social systems.

E. MILITARY APPLICATIONS OF AGENT-BASED MODELING

The Marine Corps strongly supports the application of Complexity Theory to help define and refine the Maneuver Warfare concept that is the heart of their combat philosophy.

With the help of A. Ilachinski at the Center for Naval Analysis (CNA), the Marines have developed an agent-based simulation called Irreducible Semi-Autonomous Adaptive Combat (ISAAC). [ISA97] ISAAC simulates some aspects of land warfare by using agents moving on a two dimensional grid in discrete time. Each agent bases its move decision on a rule set that includes the agent's desire to reach an objective, return to its home, confront enemy agents, stay close to friendly agents, and support wounded agents. The strength of each rule is pre-specified and can be changed during the simulation to reflect a change in behavior in response to injury.

Although this rule set is more sophisticated than the one used in SInBaD (SInBaD's rule set is discussed in chapter 3), it is still a fairly simple rule set and runs very fast in simulation. ISAAC has demonstrated that it can represent emergent battlefield patterns, such as aligning of troops and defending goals, which closely resemble combat actions used in real land warfare.

ISAAC also has the ability to pass on sensor information from one agent to another. By manipulating this parameter, G. Horne at the Marine Corps Combat Development Center used ISAAC to explore the concept of trust on the battlefield. [HOR97] The research showed a significant increase in combat effectiveness when the number of agents passing information to a deciding agent was increased.

EINSTein (Enhanced ISAAC Neural Simulation Tool) is replacing ISAAC. Its is pushing the boundary for agent-based combat modeling. Improvements to SinBad, listed in Chapter 5, include many of the enhancements incorporated in EINSTein.

The Marine Corps' work with ISAAC was the inspiration for SInBaD. Unfortunately, ISAAC does not provide the information parameters necessary to model the effects of degraded information on the battlefield, so a new simulation is needed for this research. There was also a philosophical disagreement with the agent logic used in ISAAC. ISAAC agents make decisions based on a weighting of several different factors. The agent processes those factors essentially in parallel. Current research on decision making shows that most decisions are serial in nature. This is especially true when the agent is given a limited amount of time in which to make a decision. [KLE89]

Other agent-based modeling work being supported by the Marine Corps include the simulation of the Hunter-Warrior exercise. Modelers at the Science Applications International Corporation (SAIC) are using the SWARM simulation system, developed by the Santa Fe Institute, to build a detailed model of the exercise that took place in Twenty-nine Palms in 1997. Results are still pending but they are expected to further validate the use of this technique in simulating diverse groups of forces spread out over a large region.

Most work to date has been suggestive in nature. It has helped to establish causal connections between certain measurements and battle outcome, connections that have proved elusive to validate up to this point.

III. DETAILS OF SINBAD

A. OVERVIEW

SInBad is a discrete-event, agent-based model of combat. Although it most closely represents land warfare, SInBad is an abstraction that can only suggest trends and is not intended to predict the outcome of any specific engagement. As argued in Chapter 2, this abstraction provides a powerful flexibility in the model without significantly affecting its conclusions.

This research views combat as a complex adaptive system, that is, a highly interactive system of agents that adapt to their environment. Although SInBad currently does not have the ability to simulate this adaptive nature, (this issue is addressed in chapter 5) it can still provide some insight into short-term engagements where adaptation is not as significant.

The interactive nature of combat is another matter. It is posited here that this interactivity is the cause of the rich non-linear nature of combat. In combat, agents make decisions based on their perception of the environment. Those choices affect the decisions of other agents since they make their decisions based on their environment as well. The environment becomes a constantly evolving landscape. Combat is an arena rich in complexity the progress and outcome of which can depend on small perturbations in the initial conditions. It is the understanding of this complexity that is being attempted in this thesis. Agent-based modeling of many agents acting on local information is a good technique for understanding this interaction.

Information has a major role in this interactivity. If an agent bases his decisions on inaccurate or old information, those decisions could be disastrous for the entire force. A major focus of this research is determining when accurate and timely information is important to mission success.

Developing the SInBaD simulator required the modeling of several different combat phenomena, including information, command and control, and the behavior of the combatants. The following sections describe SInBaD's basic combat environment and the logic used in simulating combat.

B. THE GENERAL SCENARIO

SInBaD's general scenario consists of two opposing forces placed on a battlefield of finite size. The combatants, represented on the display by the colors Blue and Red, have different objectives. The Blue force is an offensive force and has a goal that can be located anywhere on the battlefield. Reaching this goal is its primary objective and Blue will move toward it unless it is engaged in combat with Red forces. The Red force is a defensive force. Red agents begin the simulation by patrolling a predetermined sector between Blue and its goal. They engage Blue forces when they observe them or when fired upon. Blue and Red forces can be initially placed in up to three cells, each of any size, placed anywhere on the battlefield.

Agents progress through the simulation in discrete time steps. A time step does not inherently represent any particular unit of time. All parameters (i.e. offensive capability, defensive capability, and agent speed) are adjusted to reflect a degree of agent interaction that can be expected for a battle duration of at most a few hours. The assumption of short duration battles eliminates the need to adjust the parameters to account for the effects of fatigue and depleted munitions. Although these parameters may be important in determining the final outcome of battle, they are not considered significant in determining the relationship between the role of information and command and control structures studied in this thesis.

C. SIMULATING COMBATANTS

1. Agent Parameters

Each agent in SInBaD is an abstract center of combat power. Agent parameters, such as probability of being hit (P_h) and agent speed, can be altered to provide agents with strengths and weaknesses relative to the opposing agents in the simulation. The user is reminded that detailed, high-resolution modeling of actual forces in combat is not the purpose of SInBaD.

SInBaD is not a Cellular Automata (CA) model. CA models place agents in discrete cells and determine the value of that cell based on its surrounding cells. [BAR97:112-144] SInBaD uses an object-based model where each agent stores its

knowledge of its world in a data structure. A decision an agent makes is based on the information stored in its data structure.

Each agent has a sensor range of specified size, and views his local area with a 360 degree field of view. Agents make decisions based on what they see in this range. A circular sensor range is used in SInBaD because of its ease of implementation. For the purposes of this thesis it is not of great concern how the sensor range is modeled; what is important is that information on friendly and enemy agents in the agents local area be provided to allow a decision to be made.

An agent has an offensive capability measured by its ability to hit an opponent (P_A), and is given by:

$$P_A = \frac{P_h (1 - E_d)}{N} \qquad (3.1)$$

P_A = Probability of a hit on a single opponent in sensor range

P_h = Probability of a hit at close range with only one opponent in sensor range

E_d = Normalized distance to the enemy targeted (distance to enemy/total sensor range)

N = Other enemy located in shooters sensor range

For example, if an attacking agent has detected only one opponent and that opponent is very close to it relative to its total sensor range and its $P_h = .02$, then P_A is also equal to .02. This means that 2 percent of the time the attacking agent would score a hit against that opponent. P_A is inversely proportional to the number of enemy agents in his sensor range and decreases to 0 as the opponent reaches the limit of the attacking agent's sensor range.

Being hit does not necessarily mean being killed. Agents have a defensive capability parameter that specifies the number of hits an agent can take before being removed from play. This separation of defensive and offensive capabilities allows for the investigation of trading one for the other under specified constraints.

2. Agent Logic

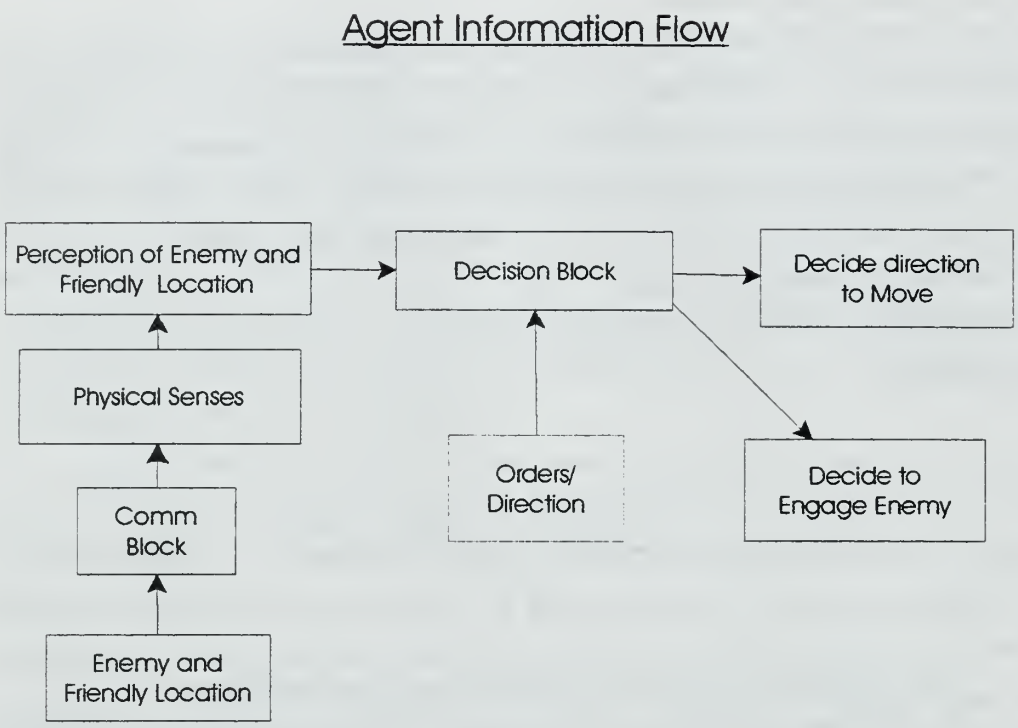

Figure 1 Agent Logic Chart

The type of combat envisioned for simulation when SInBaD was designed resembles high-paced land warfare. There exists a large body of work on decision-makers under these conditions and that thinking was incorporated into the design of SInBaD agents (this includes research on the Object-Orient-Decide-Act (OODA) loop and intuitive decision making [MCD96]). Combatants in the heat of battle make decisions under intense time pressures. High-stress decision-making in a compressed time frame forces agents to rely heavily on prior training. They also are restricted in the amount of information that is readily available to them. SInBaD agents are a simple implementation that incorporates these concepts.

Two conceptual models help define the decision-making process used in SInBaD. One model represents the flow of information. As depicted in Figure 1, information is provided to an agent either through his physical senses or by some sort of communications device. Each agent stores the location of enemy and friendly forces in his sensor range in a data structure that only he has access to. In order to model bounded rationality, the data structure has a finite size and can only store the locations of 10

enemy and 10 friendly forces. Once the information is received, the agent, in accordance with his predefined orders, makes a move and/or shoot decision based on the information stored in his data structure.

This process can also be viewed in terms of a decision-making cycle known as the OODA loop (Figure 2). For every discrete time step, the agent proceeds through this loop collecting and processing information on his local environment. He then makes a decision to move and/or shoot, and then performs the action.

Each agent performs two actions: shoot and move. Any time an enemy agent is in his sensor range, he will try to shoot it. Whether or not a hit is scored depends on his P_A. An agent also makes one of four move decisions: advance toward the enemy, hold his ground, run away, or head for a pre-specified goal.

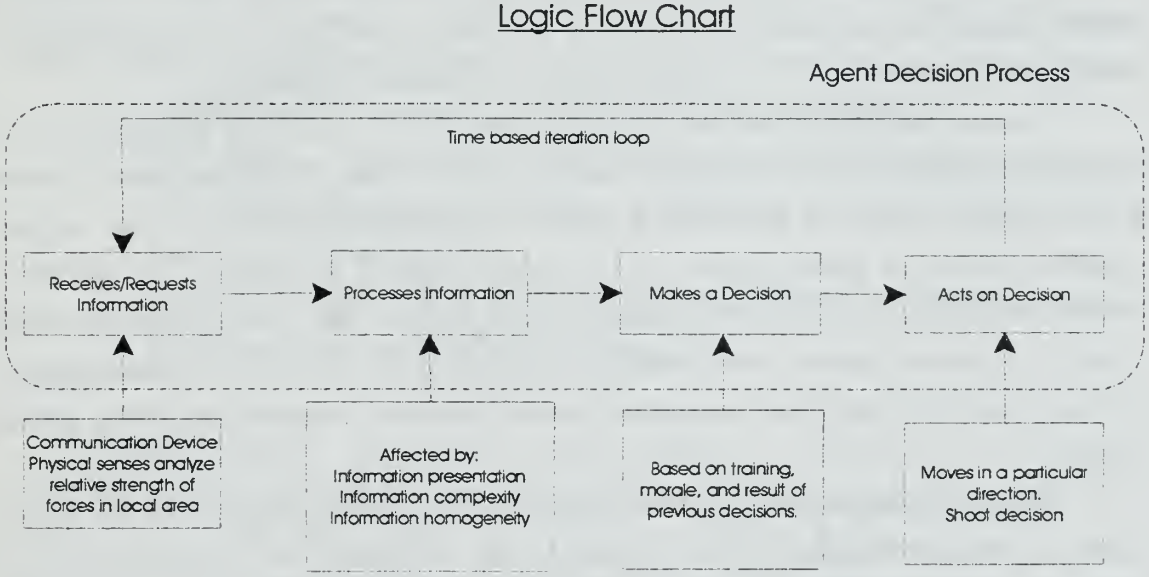

Figure 2 Agent's Logic Flow Chart

This decision is based on what he sees at that point in the simulation. In the "Receive/Request Information" step (Figure 2), the agent views his sensor range and gathers information on the number and position of enemy and friendly forces. The agent computes an *Enemy-to-Friendly* force ratio using equation 3.2, and performs one of four actions:

Enemy-to-Friendly Ratio = [Number of Enemy Forces/Number of Friendly Forces] **(3.2)**

1. If the agent's enemy-to-friendly ratio is below an advance threshold value (charge ratio), the agent will advance toward the average location of the observed enemy.
2. If the agent's enemy-to-friendly ratio is above a retreat threshold value (runaway ratio), he will retreat from the average location of the observed enemy.
3. If the agent's enemy-to-friendly ratio is between the two thresholds, he will hold his ground.
4. If the agent does not see any enemy agents, he proceeds toward the goal.

If a Blue agent does not see any Red agents, it proceeds to the goal. A Red agent initially patrols its pre-specified cell until it encounters Blue forces. If it loses track of Blue agents, it will loiter in the location where it lost track of Blue agents.

There is an *affinity rule* that overrides rules 1-3. It is a useful parameter that is used to represent a level of coordination among agents. If the affinity parameter is set to a value greater than 1, the agent will not follow the above rules until he "sees" the pre-specified number of friendly agents in his sensor range. If he does not, he will move toward the average location of the friendly agents he does see. This parameter has the effect of preventing agents from running off on their own. As will be demonstrated in the next chapter, a surprising amount of realistic behavior emerged from these simple rules.

Another important benefit of this approach is the ability to incorporate *soft factors* such as morale, fighting spirit, aggressiveness, discipline, and training into the simulation. The charge ratio (rule 1) is used in this simulation as a surrogate for morale and aggressiveness since it reflects an agent's willingness to attack based on an assessment of his side's ability to fight. The runaway ratio (rule 2) was used as a measure of morale and also training. It is reasonable to assume that a troop's decision to retreat based on the proportion of enemy to friendly forces is directly proportional to the quality and discipline of that force.

Blue forces differ from Red forces only in that they have a pre-specified goal they must attain. This makes Blue an inherently offensive force and allows the simulator to probe the advantages of offense over defense. Blue and Red force rule sets are identical once agents have engaged in combat.

D. THE ROLE OF INFORMATION

Since the main focus of this thesis is to identify the value of information in combat, how it is simulated is of great importance. SInBaD is a relatively simple simulation and information passing is not simulated in great detail. It is believed that the results generated when using abstract surrogate parameters as opposed to highly detailed models of information can still suggest trends that can be explored later with more detailed simulations. The advantage of this approach lies in its simplicity of execution and its flexibility to broadly apply the abstraction.

Information is modeled as the agent's ability to perceive his local surroundings. The presumption is that the agent has a sensor system that he relies on to produce a decision-making picture of the world. This picture is equivalent to the agent's sensor range. SInBaD allows for the adjustment of the quality of the picture an agent receives.

Information quality parameters can be divided into two categories:

1. Timeliness – a measure of how current is the information. Quality of information degrades over time. Positioning accuracy of enemy troop locations is a good example.
2. Accuracy – a measure of how precise is the information. Information that is timely is still not very useful if it is wrong.

SInBaD uses an information update parameter to adjust the timeliness of information. This parameter sets the number of steps between information updates. If it is set to a value of 10, then 10 time steps will pass before an agent receives an update of its position and the position of enemy and friendly forces in its sensor range. This means that an agent with this setting could be making decisions on information that is up to 10 time steps old.

Accuracy is modeled by introducing error in the positions of agents in the sensor range. If this parameter is set to a value of 10, then the agent will make a move decision based on the location of enemy and friendly forces (including himself) that could be in error by as much as 10 units. The error function is a uniform random distribution around the true position in both the "x" and "y" direction. This parameter allows for a separation between perception and reality; that is, what an agent thinks he sees from what is really occurring. The effect of inaccurate information on an agent's ability to hit a target is not directly modeled. It is assumed that inaccurate information leads to a reduction in an

agent's ability to hit a target. This can be accounted for directly by adjusting an agent's P_h.

It is generally believed that you can never have too much high quality information. As will be discussed in the chapter 4, this is not always the case.

E. SIMULATING A COMMAND AND CONTROL STRUCTURE

Another goal of this thesis is to evaluate the benefits of having more or less information under different types of command and control structures.

The Joint Chiefs of Staff defines Command and Control as:

> The exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communication, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. [JCS94:78]

The modeling of a command and control structure has been the most difficult aspect of the simulation and as a consequence is the area that is most abstracted. For the purposes of the SInBad simulation, Command and Control is narrowly defined to be the initial placement of troops on the battlefield and the rules of engagement (including rules of inter-agent communication) forces are expected to follow in combat.

SInBaD can model up to 100 agents on each side.[1] Each side can initially place agents into as many as three geographical locations (a total of six locations). Agents can be placed anywhere on the battlefield, and parameters of agents in that location are independently adjustable.

Different command and control structures are simulated by the initial placement of cells and the parameters used to define those cells. The decision to initially place agents in a long row, a compact ball, or break the force up to create two separate fronts

---

[1] The number of agents was limited to 200 due to the lengthy run time associated with simulating a large number of agents. Running the simulation with the 200 agent limit overtaxed the resources of the 200 Mhz Pentium machine. It is not recommended that simulations of over 40 agents on each side be used. It is hoped that later work will increase the speed of the algorithms.

are examples of actions performed by commanders that can be simulated in SInBaD. Commanders also control the mix of defensive and offensive capability provided to each area of the battle.

Of great importance here is the level of autonomy with which forces are provided. Highly autonomous agents act on their local surroundings with little regard to what is happening outside of their area of influence. SInBaD's rule set contains an "affinity rule", an overriding rule that prevents agents from acting on their environment unless they can "see" a specified number of friendly agents in a specified range. This parameter, along with the agent's sensor range, is used as a surrogate for the level of agent autonomy. An agent with a large sensor range and a desire to ensure that many friendly agents are close by defines an agent that has low autonomy, and vice-versa.

As reviewed in chapter 2, new force concepts provide more autonomy for local area commanders, a force structure that is believed to be advantageous now that current information technology can provide increased situational awareness. This hypothesis is examined in chapter 4.

IV. SIMULATION ANALYSIS

A. SIMULATION PHILOSOPHY

Before we begin the analysis of the combat scenarios modeled using SInBaD, it is important to clarify some fundamental simulation methodologies used in generating the scenarios.

1. Tuning the Simulation

A considerable point of contention in using an abstract simulation like SInBaD is deciding what initial parameter values should be used when running simulations. Since the simulation is intended only to indicate the results of a relative change in parameters and not to replicate specific battlefield conditions, a little flexibility is afforded the simulator. Still, with an arbitrarily set battle grid of 600 by 800, what does it mean for an agent to have a speed of 2? It is clear the parameters in the simulation should not be interpreted literally. Based on the literature reviews conducted there appears to be little work done on determining how to justify numerical parameter decisions. [ISA97] [AXL97] [WAL95] [SFI95]

The goal is to find a parameter space that yields results consistent with real warfare. If SInBaD agent behavior is completely irrational or unexpected for a given set of values then those values would be suspect. The approach taken for this thesis is to test a range of agent parameter values on an attrition scenario. The attrition scenario is defined as both sides having equal values for all parameters. This scenario was chosen because it gives Red and Blue identical capabilities, an important factor in ensuring that the change in behavior is due to a change in parameters and not the result of a relative change in one side's capability over the other. Another advantage to using the attrition scenario is that the behavior of real combatants in this scenario has been heavily studied. [MOR95] It can be argued that no combat scenario is well understood, but perhaps the attrition scenario comes closest.

Agent behavior for this scenario resembled behavior that would be expected in a real combat attrition scenario (the detailed analysis of this scenario is presented in the next section of this chapter). It became apparent after several changes in parameters that the agents are remarkably well behaved over a large parameter space. Their behaviors do

change, but in the direction one would expect for the change made. Response to the speed parameter is a good example. When the agents' speeds are increased from 2 to 10, their actions are more chaotic. Battle lines are not as smooth, often breaking up into separate battles. More agents sneak by enemy lines, leaving fellow forces undermanned. The behavior suggests that highly mobile troops need better coordination than slower ones to prevent them from leaving the rest of the force behind. Instead of defining a parameter space of reasonable values, the exercise only produced more questions for further research.

In the end, the decision was made to tune the parameters to a range that met the following criteria:

1. Provide a simulation long enough to allow for a large number of agent interactions. The results of the battle should be decided by the 2000^{th} time step. This was important in keeping the simulation run time short.
2. Provide a simulation where the results of actions did not occur so quickly that they could not be viewed. Although SInBaD has the capability to plot the statistical results of several runs, most of the conclusions were generated from watching the agents interact on the visual display.
3. Provide a simulation where the agent's actions are reasonably well behaved, that is, reminiscent of well-trained forces. As will be shown later, well-behaved forces in one scenario are not always well behaved when placed in a different situation.

2. Winning and Losing and Risk

Determining when a parameter set delivers a "win" for Blue or Red is not as simple as it might seem. Many engagements result in Blue breaking past Red's battle line but only after receiving heavy losses. Every situation in real combat is different and the commander must decide if the value of the objective is worth the risk of his assets.

The approach taken is to generate a statistics for value and cost. How well a force achieved their objective is measured by recording the average position of all surviving forces as a function of time. An additional metric for Blue is the tracking of surviving forces that reached their goal. Combat losses over time are measured to evaluate the cost of an action.

A commander or planner is also interested in a scenario's level of risk. Many scenarios appeared very sensitive to slightly different initial conditions, as well as small changes in the information parameters. The variation in the objective and cost measures resulting from a small perturbation in the initial conditions is used as a measurement for risk. Changes in the objective and cost measures average values resulting from a small change in the parameter space are also used as a measure of risk. The justification for the use of variation in the above measures to measure risk, and the determination of what makes a particular scenario yield high variations is the subject of the planning section in chapter 4.

The research will be presented in these terms: objectives, costs, and risks. It is left to the reader to decide if a particular action has enough value to justify the cost and the risk.

3. Battles on the Edge

As discussed earlier, the difficulty in simulating combat is that many differential-based models assume an equilibrium situation in which forces behave in a linearly scalable fashion. That is, forces can be represented by an aggregated value of a smaller subset of forces. A situation when this is a valid assumption is when the asymmetries between opposing forces is great. It is in these situations that a small perturbation in the parameter space will not lead to a disproportionately large change in the results.

These scenarios are uninteresting for two reasons:

1. Scenarios such that one side has a significant advantage tend to lead to results that are expected and are not in need of simulation.
2. The trend in combat strategy today is to leverage information to compensate for a reduction in force. This desire to improve the effectiveness of our forces so that we can operate with even fewer of them seems to be a recipe for staying far away from the equilibrium condition.

For these reasons, this thesis concentrates on scenarios where risk is great. That is, the focus will be on scenarios where small changes in force structure and capability can lead to significant changes in battle outcome.

B. VALIDATION

The physical placement of forces in most of the scenarios considered is identical. Blue forces are initially placed at the top of the screen, with a goal positioned at the bottom. Red forces are placed in between Blue's initial placement and its goal. Red initially is in a patrol mode until they see Blue. When one side sees the other, combat begins. Figure 3 shows a typical initial placement of Red and Blue forces on the battlefield.

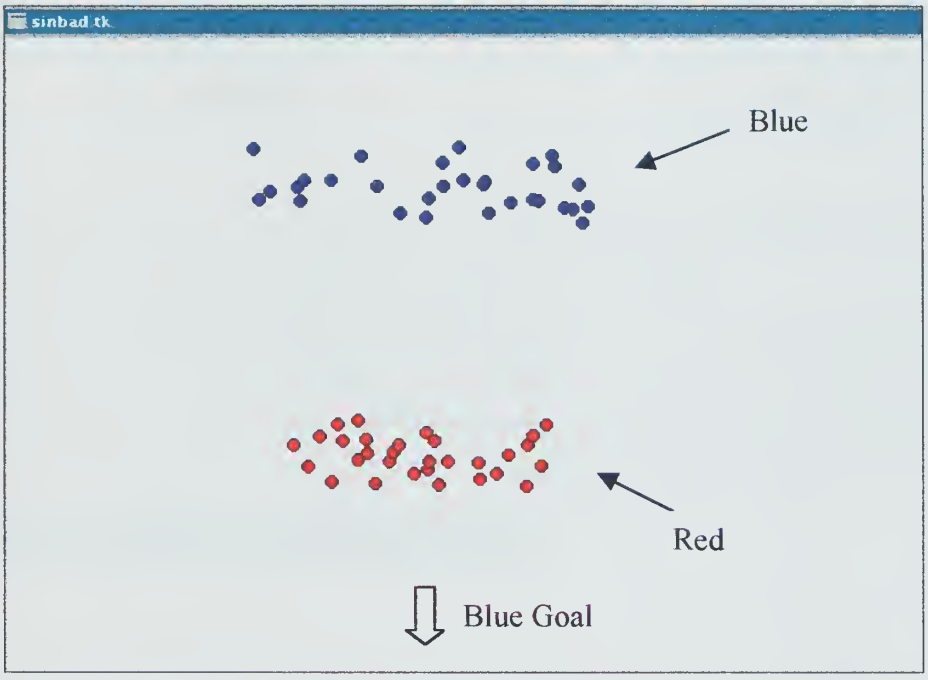

Figure 3 Attrition Scenario - Initial Placement

The first scenario analyzed is a test case to determine whether the simulation yields reasonable results under familiar combat situations. Since much combat modeling is based on the attrition warfare scenario, it is the first scenario modeled. Table 2 shows the parameters used for this simulation. Thirty agents on each side are given equal defensive and offensive capability. The only difference between the forces is Blue's desire to reach the bottom of the screen.

Table 2 Attrition Scenario

Red Forces		Blue Forces	
30	Red agents in cell 1	30	Blue agents in cell 1
300	cell 1 box center x	300	cell 1 box center x
300	cell 1 box center y	100	cell 1 box center y
300	cell 1 box size x	100	cell 1 box size x
50	cell 1 box size y	50	cell 1 box size y
N/A	cell 1 goal x	300	cell 1 goal x
N/A	cell 1 goal y	500	cell 1 goal y
.01	cell 1 pk	.01	cell 1 pk
2	cell 1 speed	2	cell 1 speed
100	cell 1 sensor/shooter range	100	cell 1 sensor/shooter range
.5	cell 1 charge ratio	.5	cell 1 charge ratio
1	cell 1 runaway ratio	1	cell 1 runaway ratio
3	cell 1 max hits	3	cell 1 max hits

The affinity parameter has been set to zero, meaning that agent actions are not concerned with the location of friendly forces prior to attacking.

As the simulation progresses, agents form into lines of attacking agents (Figure 4). Even though the agents are acting in a completely autonomous fashion using a very small set of rules, a complex and coordinated behavior results. This coordinated behavior resembles that which might be expected in an actual battlefield scenario. The resulting battle lines are reminiscent of World War I trench warfare.

Figure 4 Attrition Scenario - Active Engagement

The attrition scenario gives some confidence that realistic behaviors can occur with the simple rules used. Although this scenario is used primarily as a test case, it yielded two interesting observations, one expected and one that is a little surprising.

The expected result is that neither force consistently obtains it's objective. Figure 5 shows the average Y location of the surviving forces as a function of time. The Y location is an indicator of how far forces have progressed in the direction of interest (the vertical direction).

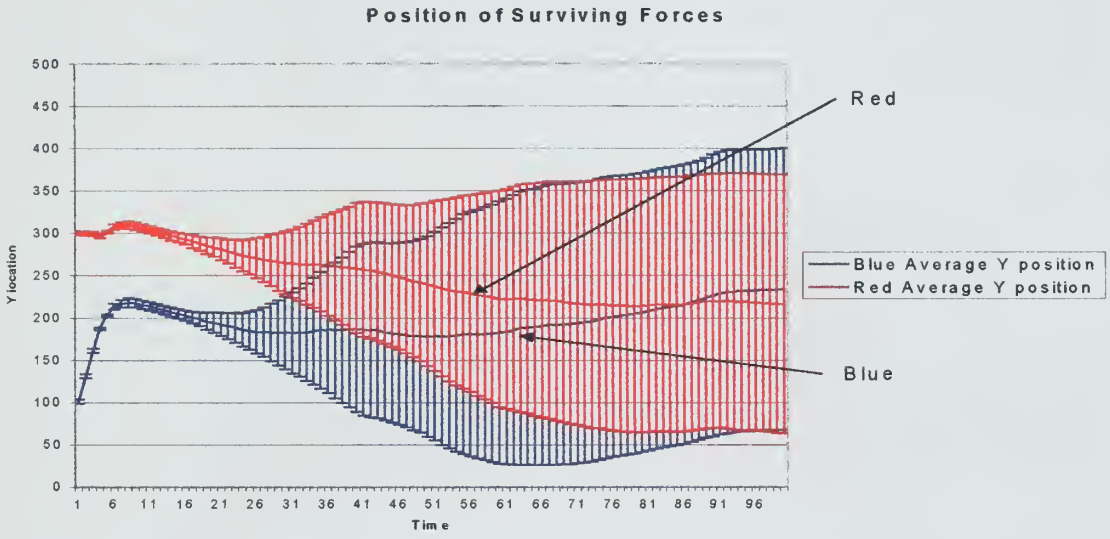

Figure 5 Force Location Attrition Scenario

All time-based graphs plot the results of 10 simulations each using a different random seed. Bars on the graph reflect one standard deviation from the average. The large deviation from the average indicates that neither force achieved its objective with any degree of consistency.

Figure 6 shows the losses each side suffered over time using the same simulation. It could be argued from the results that it is better to be on defense rather than offense since Blue casualties tend to be a bit higher. Clausewitz would certainly agree. He argued in his classic text "On War" that, for a variety of reasons, the defensive position is preferable.

In SInBaD, the causality of the result is buried deep in the interaction of the agents and probably stems from the underlying driving pressure Blue forces feel because of their ever-present goal. This subtle pressure is enough to place Blue forces in an advancing position; otherwise they might have fled.

The more surprising result, especially considering the small set of rules, is that forces do not fight to the last man. Most battles resulted in a turning point where one side or the other decided to exodus *en masse*. This result is frequently seen in real combat. With few exceptions, combatants do not fight to the last man. In reality, one side will receive a certain number of casualties and then flee before reaching complete annihilation. What factors lead to this turning point is a subject of hot debate. It has been argued that training, moral, and even the rate of advance of the superior force are critical factors.

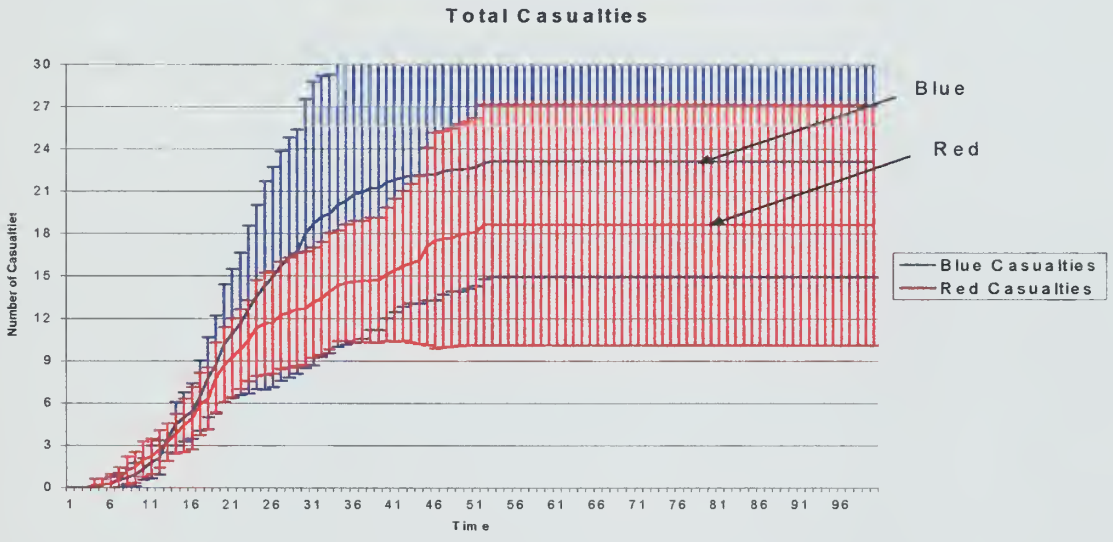

Figure 6 Attrition Scenario - Agent Losses Averaged Over 10 Runs

The explanation in SInBaD is again the result of agent interaction. There is no master agent to call for a retreat when overall force structure falls below a specified threshold. An agent in this scenario is acting with complete autonomy based on observations of his local environment. Recall that an agent will retreat when the enemy-to-friendly ratio reaches a certain limit. By chance, an area of the battle will reach this limit for agents in that local area. Once they begin to retreat, neighboring forces see a

change in the force structure in their sensor range and begin to retreat as well. The result is a cascading effect that usually results in a complete force withdrawal. The retreat may be permanent but sometimes a retreating force will rally and hold their ground. With chance on their side, the underdog force can destroy enough enemy forces to begin another advance and successfully obtain its ultimate objective.

This is an excellent demonstration of a feature of agent-based simulation. With opposing forces so equally matched, small events that at the time seem inconsequential, can significantly affect the course of battle. These results could only be demonstrated using agent-based modeling.

It is easy to see that using an agent-based simulation, like SInBaD, could suggest what factors might cause a hasty retreat. Unfortunately, that research is beyond the scope of this thesis.

C. ANALYZING THE EFFECTS OF INFORMATION

1. High Tech vs. Numerically Superior

A primary focus of this research is exploring the relationship between information and command structure. Proposed command structures such as those discussed in the previous section are characterized by more autonomous control where commanders in the field make the decisions. How much information should a local commander have, and in what circumstances does this information benefit him?

We begin by defining a scenario. Again the battlefield is set up with Red forces in the direct path of Blue's initial placement and its physical goal. Keeping with proposed force structures, it is assumed that Blue is a more technologically advanced force. It will have better communications and more lethality to compensate for a numerical disadvantage. Red is a traditional force represented by the base case used in the validation. One significant change for Red is that its affinity parameter was set to 5 to reflect a less autonomous command structure. The parameters used for the initial run are listed in Table 3.

Blue force size is significantly the smaller (2/3 that of Red). To compensate for this, Blue is given greater lethality (.03 P_h vs. .02 P_h for Red) and better mobility (4 vs. 2 in the speed parameter). Blue's runaway ratio has been increased to reflect better

training. These parameters represent a modern, well-trained, highly mobile autonomous force (High Tech) fighting a more traditionally controlled force (Numerically Superior).

The results (Figure 7) show that Blue's added lethality and mobility are not significant enough to overcome Red's numerical advantage. Red consistently prevents Blue from achieving its objective of reaching the Y location of 500. Blue also sustained a significant number of casualties, losing approximately 15 of its 20 original agents by the end of the simulation (Figure 8).

The question now pondered is: does increased local awareness (through an increase in sensor range) improve Blue's situation? We call the above scenario case 1. Case 2 only differs from Case 1 by increasing the size of Blue's sensor range. The first attempt to give Blue an edge through information is to increase the its sensor range to 150, which corresponds to a 3 to 2 advantage in that parameter. As Figure 10 suggests, Blue does perform better with the increased sensor range but not exceedingly so. On average, they are more likely to achieve their goal and lose fewer forces in doing so (Figure 11).

Table 3 High Tech (Blue) vs. Numerically Superior (Red) (Case 1)

Red Forces		Blue Forces	
30	Red agents in cell 1	20	Blue agents in cell 1
300	cell 1 box center x	300	cell 1 box center x
300	cell 1 box center y	100	cell 1 box center y
500	cell 1 box size x	500	cell 1 box size x
50	cell 1 box size y	50	cell 1 box size y
N/A	cell 1 goal x	300	cell 1 goal x
N/A	cell 1 goal y	500	cell 1 goal y
.02	cell 1 P_h	.03	cell 1 P_h
2	cell 1 speed	4	cell 1 speed
100	cell 1 sensor/shooter range	100	cell 1 sensor/shooter range
.5	cell 1 charge ratio	.5	cell 1 charge ratio
1	cell 1 runaway ratio	1.5	cell 1 runaway ratio
3	cell 1 max hits	3	cell 1 max hits
5	affinity parameter	0	affinity parameter

Figure 7 Force Location for High Tech (Blue) vs. Numerically Superior (Red) - Case 1

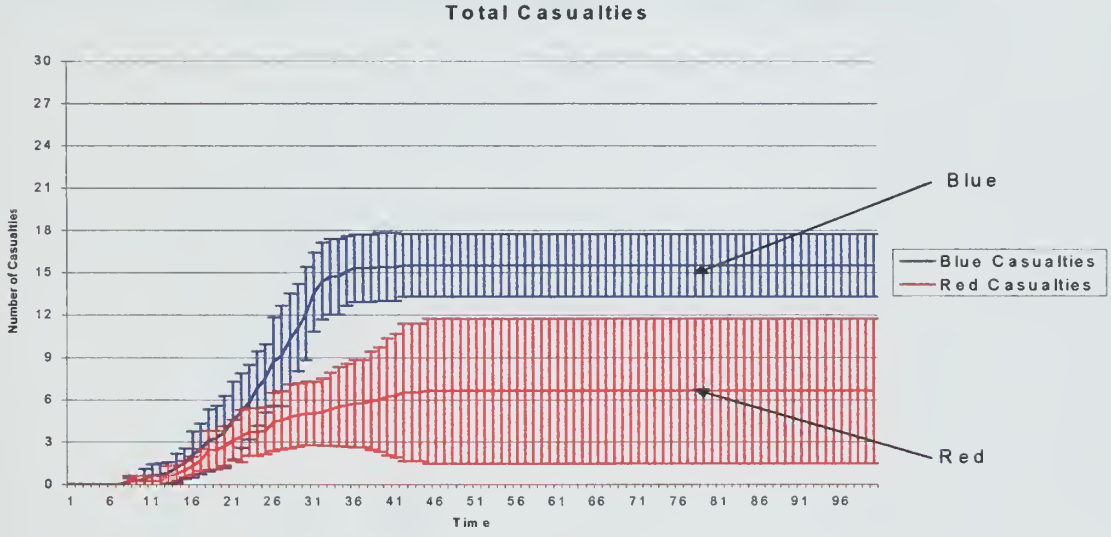

Figure 8 Casualties for High Tech (Blue) verses Numerically Superior (Red) Case 1

What is it about an increase in sensor range that is helping Blue meet its objective? Why isn't Blue's increased sensor range providing more of a benefit? Viewing the agents in battle is necessary to understand the dynamics of the situation. In Case 1, Blue and Red forces have the same sensor range and they begin shooting at the same

time. Traditional battle lines are drawn similar to that seen in the attrition scenario. Blue's greater speed is not much of an advantage here since their forward progress is halted by the interaction with Red.

With increased sensor range, Blue sees Red first and begins shooting first. The range between 100 and 150 units is critical since, in this range, Red becomes the proverbial "fish in a barrel". Red cannot see Blue when they are within this range band. One would suspect that this scenario has all the makings of a route favoring Blue, and if this condition were to remain throughout the simulation the battle would be over with the complete destruction of Red forces.

Yet, when analyzing force losses for the Case 2 – 150, Red actually sustains comparatively smaller losses over the Case 1 scenario, especially considering their larger initial force (Figure 11). Blue's progress toward its goal, although better, is not ideal. Recall that Blue's objective is to reach 500 units in the Y direction.

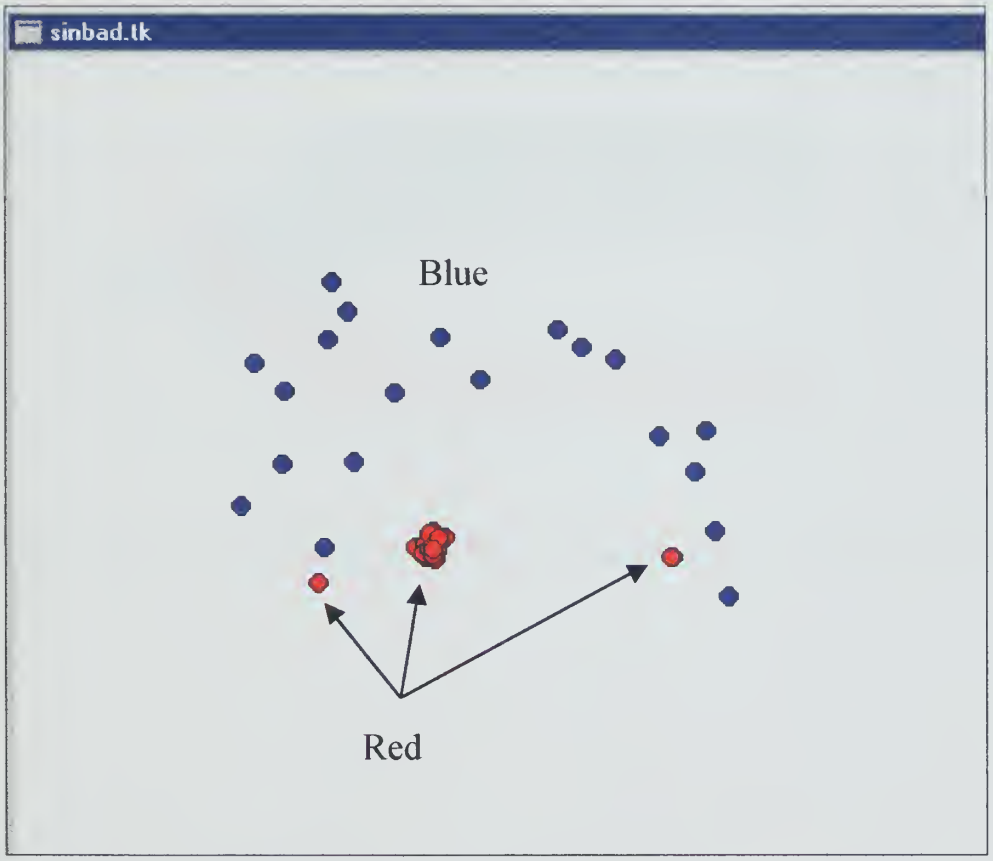

Figure 9 Red Herding Behavior Case 2 - 150

Red was able to get out of the proverbial barrel by adapting its behavior. In every run studied, Red adopts a herding tactic, in which all the agents clump together into one or two large groups (Figure 9). It is not quite clear why Red takes on this new herding behavior. The affinity parameter was the first suspect since this was a measure of increased coordination between the agents. When the parameter was set to 0, herding still occurred but not as efficiently.

The tactic seems counter-intuitive at first since concentrating forces provides Blue with one big target to attack versus several smaller targets that would seem more difficult to hit. But concentrating forces gives Red an advantage by improving Red agents' enemy-to-friendly force ratio in their local area. Since that force ratio is the key parameter that allows agents to advance on the enemy, Red became the aggressor, dissecting Blue forces and attacking smaller groups of agents in series.

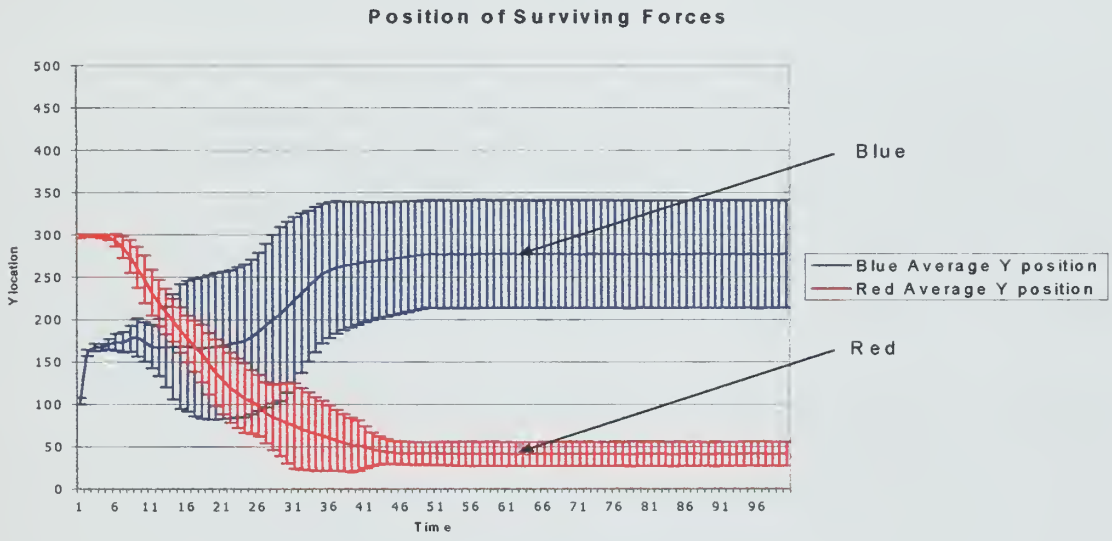

Figure 10 Force Location for High Tech (Blue) vs. Numerically Superior (Red) – Case 2 (150)

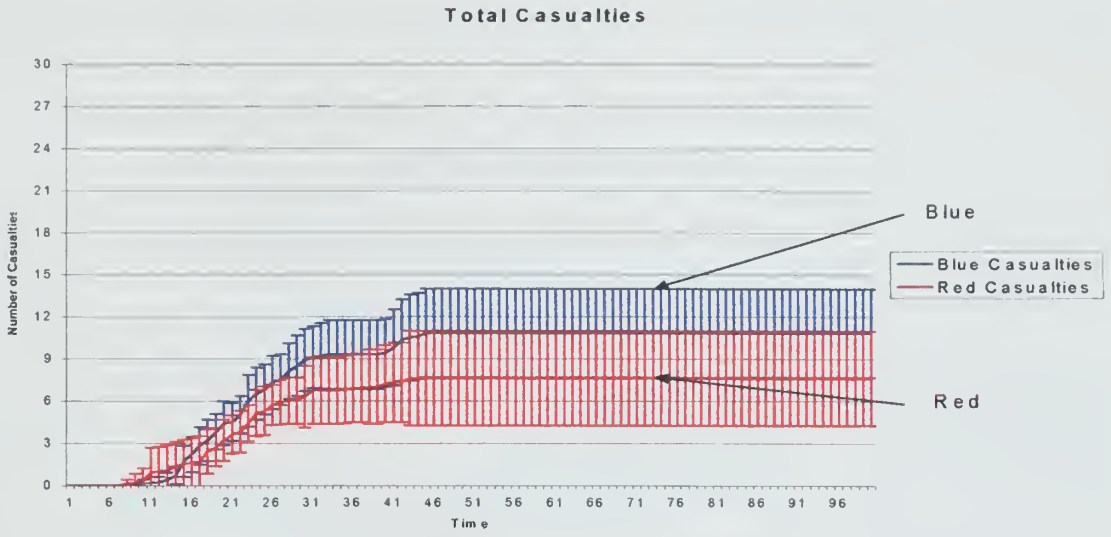

Figure 11 Casualties for High Tech (Blue) vs. Numerically Superior (Red) Case 2 – 150

Blue had the early advantage when it is shooting in the 100 to 150 range but this is countered by Red's herding tactic. The disadvantage of Red's herding tactic is the establishment of large gaps in Red's defenses that allows Blue to slip by Red and accounts for Blue's improved progress toward his goal (Figure 13). Some forces are able to slip by but on average they take horrendous losses because of Red's overwhelming numerical superiority.

If increasing the sensor range to 150 is good; certainly increasing it to 200 should be better, but this does not seem to be the case. Figure 12 shows Blue's progress toward the goal with a sensor range of 200. Recall that a Blue agent's ability to shoot Red is a function of the number of agents he encounters. With such a large sensor range, Blue's ability to inflict much damage on a particular Red is reduced. Red still employs its herding tactic and is able to get in close to Blue agents, preventing them from achieving their goal.

2. High Tech vs. Numerically Superior – Conclusion

Figure 13 summarizes the 10 run average results of the three scenarios with regards to Blue; the standard deviation bars were removed for clarity. The figures represent Blue's progress for all three cases as a function of time. Numbers in the boxes

represent the average number of Blue agents that have reached the goal by the end of the simulation.

Figure 12 Force Location for High Tech (Blue) vs. Numerically Superior (Red) – Case 2 (200)

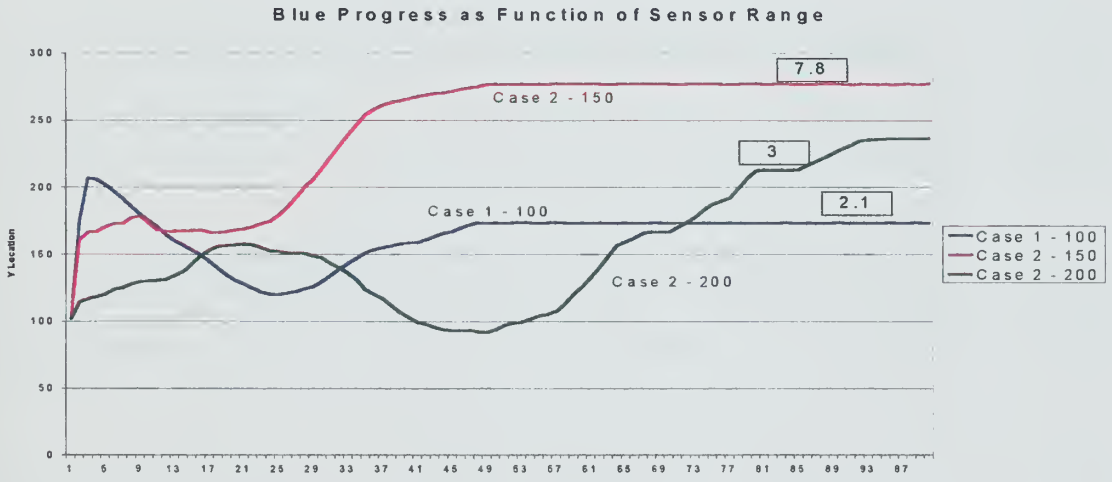

Figure 13 Blue's Progress with Varying Sensor Range

Case 1, the sensor parity case, provides a clear advantage for Red. Blue makes good progress in the beginning only because it does not see Red until it reaches 200 units.

Once combat begins, Blue is repelled and takes heavy losses. On average, only 2.1 Blue agents are alive and at the goal by the end of the simulation.

Case 2 (150) provides Blue its best performance against Red, both in attaining its objective (7.8 of a total of 20 agents reach the goal), and reducing its losses.

Case 2 (200) shows a drop in performance in attaining its objective. Blue casualties are light at first due to the larger initial distance from Red. On average, only 3 agents are at the goal by the end of the simulation. Red needs more time to reach a range at which it can attack Blue, but eventually does and disrupts Blue's plans.

The results in this section provide some insight for actual combat. In combat, more information has usually been considered advantageous but this scenario suggests that more information can lead to a less than optimal application of force. Although it is better to have more information than your opponent, there is a limit to the amount of information a field commander can use. The number of targets a combatant can effectively manage determines that limit. Recall that the agents in SInBaD are rationally bounded and can only effectively process a limited number of targets (value is set to 10). Although the exact interplay between bounded rationality and sub-optimal target selection is difficult to assess given the analysis tools currently provided in SInBaD, the data suggests that bounded rationality affects two important issues.

A commander in the field making targeting decisions should have some priority scheme. SInBaD agents do not have a targeting prioritizing algorithm. This, coupled with their long sensor range contributes to poor target selection. Targets with low probability of being hit (i.e. very far away) are just as likely to be targeted as ones close by.

Also, a commander should not be assigned more targets than he can cope with. When more enemy targets are in range than can be effectively managed, consistency in the application of force is lost. This is representative of an information overload condition.

The challenge is to find the information balancing point; too little information deprives the commander of an information advantage, too much can lead to information overload and poor decision making.

Another interesting result stems from Red's behavior in dealing with Blue's increased sensor capability. Red's actions suggest that herding into a powerful point of combat power may be an effective counter to a technologically superior force. This tactic gives Red agents a sense of physical superiority in their local area and allows them

to "divide and conquer" the Blue force by focusing on small segments of Blue's line of battle.

The data provides a caution for an autonomous force. Blue's lack of coordination resulting from agent autonomy allowed it to be dissected and sequentially attacked. It would appear that some level of coordination is necessary to prevent this from happening.

If the future brings more autonomous command and control structures to the U.S. military, an emphasis must be placed on providing the right balance of information to the local commander and the right balance of coordination of all the local commanders.

One variation in the scenario, which was not discussed above, is worth mentioning. That particular scenario reduces each Blue's sensor range to a point where all Blue agents are essentially blind. The reduced sensor range causes Blue to ignore Red as it proceeds toward the goal. The scenario is similar to a Blitzkrieg or lightning charge, where the goal is not to engage the enemy but run past them. This can be a very effective tactic only if Blue is significantly faster than Red, and the lethality of Red against Blue is not great. Otherwise, it is suicide.

3. Application of Information Warfare

Results from the previous section provide clues to better preparing a smaller high technology force by focusing on coordination to optimally apply firepower. Still, Blue's performance with respect to obtaining its objective and reducing casualties could surely stand improvement.

Information Warfare (IW) has been touted as a force multiplier. Proponents of IW claim that its application provides a significant advantage by slowing down an opponent's decision cycle (OODA loop). It is argued that the force with the faster decision-making process can control the events of a battle, forcing an opponent to "react" to your decisions rather than act on his pre-established plan.

To test these assertions using SInBaD, the quality and timeliness of information (defined in Chapter 3) received by Red agents is adjusted. It is not the concern of this research to determine the "how" of applying IW, the focus here is on the effects of successfully applying IW. It is assumed that Blue possesses the capability to effectively degrade Red's information.

The agent parameters for both Red and Blue remain unchanged from Case 2 (150) (see Table 3). The change is in the two parameters used to control information flow: the update parameter and the position error. Recall that the update parameter determines the number of time steps that pass before an agent receives an update on the position of enemy and friendly forces in his sensor range. Position error determines the accuracy of information provided to an agent for the purposes of making a move decision and is measured in distance units.

Figure 14 and Figure 15 (compare with Figure 10 and Figure 11) show the results with the information update and position error both set to a value of 20. It appears that the application of IW is of little value. Blue's progress toward his objective is actually impeded, not improved by its use of IW, and Blue still loses about half his force. Does this mean that IW does not provide any benefit to the warfighter? Hardly: as with all the previous studies, the time plots are helpful in determining if a set of parameters cause the agents to achieve their objective but provide little insight into the dynamics that produce the results.

Figure 14 Force Location for Application of IW - Case 1

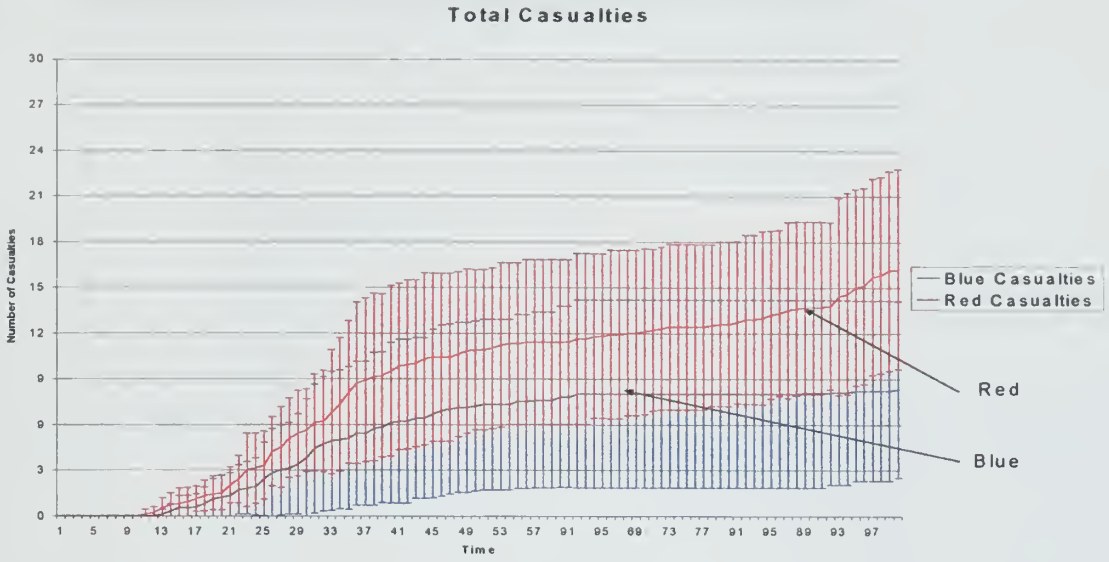

Figure 15 Casualties for Application of IW Case 1

When the individual runs are observed a pattern emerges that explains why IW is not a successful tactic in this scenario. As in the High Tech vs. Superior Numbers scenario above, Red agents herd into a few points of combat power (see Figure 9) and effectively stop Blue's advance toward his objective. Once Red has halted Blue's advance, the battle takes on a static nature where little movement is witnessed over several hundred time steps. It is this static nature that makes IW ineffective. When the change in the position of forces is small, IW has less effect because information timeliness is less critical. Red forces are fed information that can be as much as 20 time steps old, but Blue agents have not moved very far from their last known location so the information still has some value.

If Blue is ever to break past Red, it must find a way to increase the dynamic nature of battle. Several strategies have been tried including separating Blue's force into two smaller forces, and attacking at different times, but the most effective strategy is to increase Blue's aggressive nature. The attack ratio is increased to one, meaning that Blue agents advance whenever they see an enemy-to-friendly ratio of one or less. Recall in the previous scenario that a Blue agent must see at least twice as many Blues as Reds before it advances. The new scenario also includes Blues use of attacking from two distinct locations, effectively adding to the chaotic nature of battle.

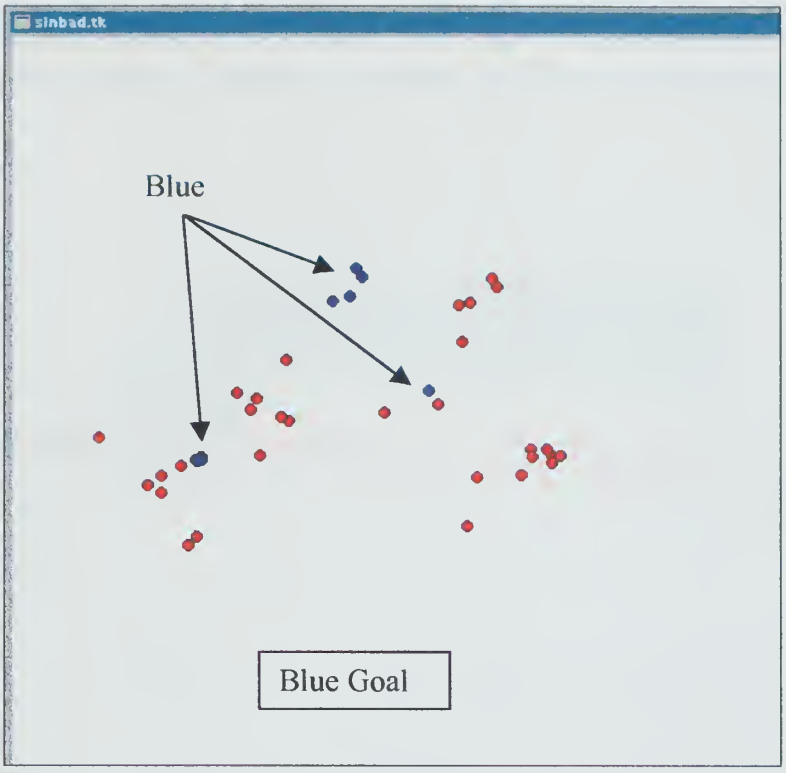

Figure 16 IW Scenario with Relatively Aggressive Blue Agents

The effect of both of these actions on the dynamic nature of battle is shown in Figure 16 (compare with Figure 9). The new battle conditions are more chaotic. Red is scattered over the entire battlefield by Blue's aggressive nature. With Blue's greater speed, Red agents race to a location only to find no one there. This tactic effectively takes those agents out of play. With Red's force dispersed, Blue can now attack smaller groups while leaving Red's disoriented and ineffective forces behind.

Figure 17 and Figure 18 show the results using the new attack ratio and two cells approaching from different locations. Blue's progress toward the goal is only slightly impeded around time steps 6 to 21. Although Red's forces are scattered all over the field, their average position remains approximately where it began, a testimonial to Red's lack of effectiveness. The application of IW when the dynamic nature of battle is kept high allows the Blue force to consistently obtain its objective with reduced casualties. Even though Red has a significantly higher level of forces (a 3 to 2 favorable force ratio) they are rendered ineffective and take on heavy casualties. Recall that the parameters used to simulate IW do not affect an agent's ability to hit a target in his sensor range, only the

agent's decision on where to move is affected. It is reasonable to assume that Red's ability to attack Blue would also be diminished, further compounding Red's problems.

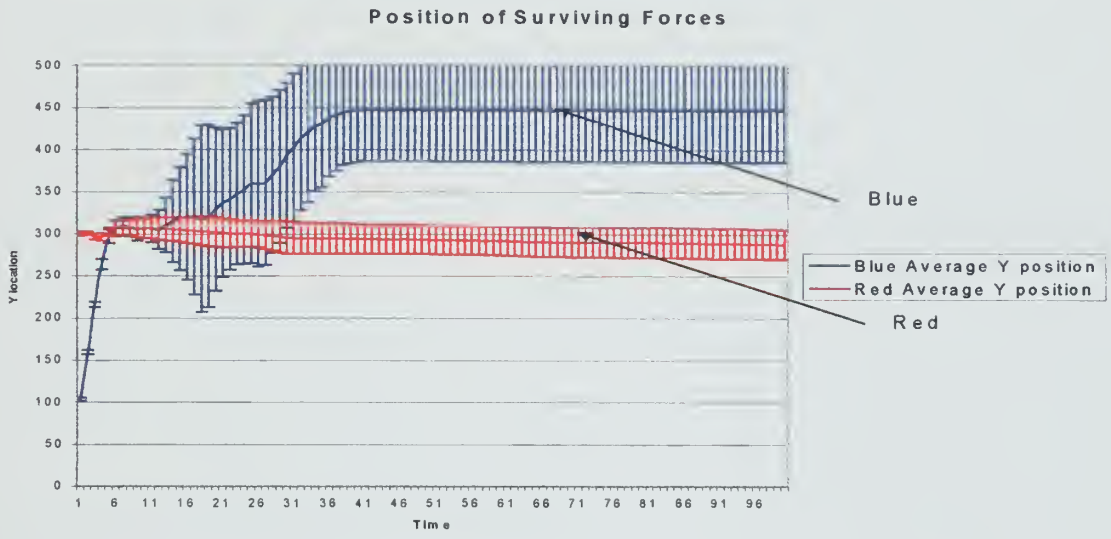

Figure 17 Force Location for IW Scenario with Aggressive Blue Agents

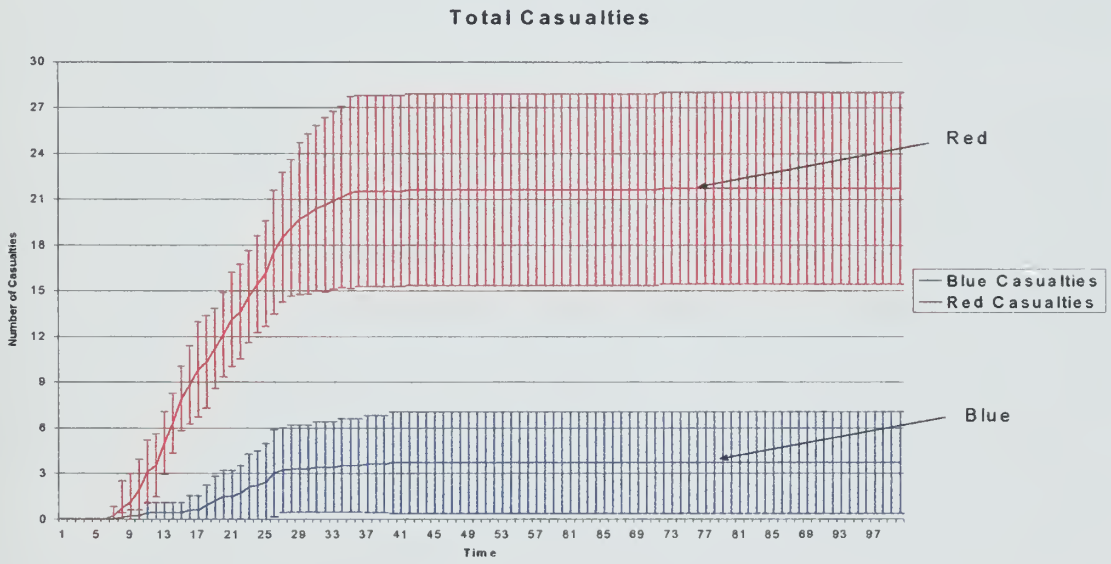

Figure 18 Casualties for IW Scenario with Aggressive Blue Agents

4. Application of Information Warfare – Conclusion

Figure 19 Force Location Summary of IW Scenario

This IW scenario shows two important results. The first relates to the timeliness of information and its relation to IW. The second result shows how to make information more time sensitive.

The results of the previous section (Figure 19) suggest that the effectiveness of IW is a function of the time-dependent nature of the information under attack. Combat situations where the value of information decreases slowly over time is less affected by delays. In other words, information can be thought of as having a characteristic time length. Even if an IW attack degrades an information flow, if updates stay within this characteristic time length, the attack will not be effective. When conducting IW planning, the focus should be either on attacking sources of information with a short characteristic time length, or in finding ways to make the situation more time sensitive.

Making a situation more time sensitive is aptly illustrated in the IW scenario above. By aggressively charging Red's position, Blue is able to significantly reduce the amount of time Red's information held value. Blue keeps Red on the move, never letting battle positions stagnate. In order for Red to stay informed on the rapidly changing situation, information updates must occur more frequently. Unable to get the

information it needs, Red begins to lose situational awareness. The situation worsens until Red can no longer track Blue forces.

The lesson here is that an effective way to make information resources more time sensitive is to increase the dynamic nature of battle. Neat, predictable, battle lines are easier to control than a chaotic, messy battlefield where forces are dispersed over a wide area. New information technologies can allow forces to operate effectively on a chaotic battlefield, but the data suggests that this force is heavily reliant on information to be timely and accurate. If not, the valuable asset known as Information Technology can turn into a disastrous liability.

D. A PLANNING AND REAL-TIME ANALYSIS TOOL

In the military arena, agent-based simulation is still in its infancy. Although this type of simulation has been used to "suggest" possible cause-and-effect relationships that lead to certain battlefield results, the real goal is to provide a useful tool to the warfighter. During the course of this research, it became clear that simulations like SInBaD had the potential to be a useful aid to combat decision making, both at the pre-planning stage and in real-time to assist a commander actively engaged in battle.

SInBaD is currently too abstract and simple to serve this purpose, it is hoped that more sophisticated progeny will be developed that can validate agent-based simulation's potential as a combat decision aid. For the purpose of discussion, the applicability of agent-based simulation as a decision aid is divided into 3 categories:

1. Combat Strategy Development

The scenarios discussed in this thesis suggest that the dynamic nature of the battlefield can profoundly influence the information requirements of the fighting forces. A fast, autonomous force can be effective provided is has the ability to keep the chaotic nature of the battlefield high. It was also suggested that the application of information warfare is more effective in the chaotic battlefield. These are just a few insights that can be culled together to help develop a combat strategy for information age warfare. Running more complex and realistic simulations can help identify parameters that affect the dynamics of a situation, and thus the battle outcome.

This type of simulation can also be used to analyze many parameters other than those related to information. The Marine Corps Combat Development Command has

just such a goal with their Project Albert. Using the ISAAC simulation discussed earlier, they are developing a multi-dimensional parameter space that can tie several parameters together. A. Brandstein at MCCDC calls this technique "Data Farming". [HOR98] He realized that data collected from actual combat experience is rare and questionable in content when considering the extreme situations the data collectors are under. In order to establish a new combat strategy, many more data points are needed. This is the same problem that drove Tom Ray to create Tierra (chapter 2). Brandstein found his artificial world in ISAAC, the simulation discussed earlier. By incrementally changing the over 50 parameters built into ISAAC, a fitness landscape can be generated that could provide new insights like those mentioned in this research. As these computer simulations become more complex, it is believed that it can become a valuable tool in developing a coherent combat strategy and provide future military leaders with tools they need to make decisions on strategy and acquisition.

2. Identifying Risk and Critical Linkages

A large number of factors, such as the number of troops, must be considered when a combat plan is developed. Considering the number of variables that are analyzed, it would be of benefit to have a planning tool that can identify which variables increase risk if their values deviate slightly from the plan. If a battle plan is simulated on a computer, it may be able to identify high-risk parameters. This risk may stem from any number of factors including not having enough forces or other critical materials in a given area to accomplish the mission. *Error analysis* is the tool used here to identify these high-risk situations.

For this discussion, error analysis is defined as the process of measuring risk based on the size of the variation in the simulation results. As an example of how error analysis helps identify risk, let's assume that two scenarios provide a favorable final location for Blue's average force. One scenario achieves that quite frequently as evidenced by a small variation between runs. The other scenario also does well on average, but the variation of the results is high. The conclusion is that the scenario with high variation has more risk since the objective cannot be consistently met.

The latter scenario can also be characterized as highly non-linear. The term "non-linear" has been bandied about in the current military literature to the point where its definition has become vague. At the risk of muddying the waters further, it will be used

here to define those scenarios that result in high variation in the mission objectives. For it is these scenarios that are highly sensitive to small perturbations in the initial conditions. These small perturbations are difficult to impossible to control. Attempting to find the causality that leads to poor results within a single run is futile because it is the result of the complex interaction between the agents in the simulation. In a highly non-linear scenario, infinitesimally small deviations in the initial conditions can produce surprisingly different outcomes. In order to reduce the risk, the initial conditions themselves must be changed. If a campaign planner can identify when a planned engagement exhibits this risky variation, steps can be taken to alter the scenario.

Another source of risk is *critical linkages.* A critical linkage defines a connection between two missions that may prove critical to the success of the overall mission. The analysis of critical linkages requires a simulation system that can model the hierarchical nature of combat. For example, suppose a campaign plan consists of separate and nearly simultaneous battles occurring over a large land region. This scenario is representative of those discussed in the Ship-to-Objective Maneuver concept in chapter 2. Each tactical battle can be simulated along with the interaction between battles. This multi-layer simulation can help determine *four-dimensional centers of gravity.* These centers of gravity are points in the battle where the success of one portion of the campaign plan becomes critical to the overall mission's success. They are four-dimensional because specific points in the battle may only be critical for a period of time. Good commanders have relied on intuition to identify these four-dimensional centers of gravity. Agent-based simulation could become a powerful assistant to help reduce risk by identifying highly non-linear scenarios and critical linkages.

3. Real-Time Analysis

The battlefield of the future will be a complex one. Commanders will need to assess an overwhelming amount of information, reach conclusions, and make decisions in a compressed time span. Agent-based simulation could provide valuable assistance by recommending courses of action and identifying four-dimensional centers of gravity in real time. Imagine a powerful computer tied directly to sensors that provide current enemy and friendly troop positions along with other combat parameters. This computer is continually running simulations using the updated sensor information to predict possible courses of actions and resultant outcomes. These computers can not take the

place of a human commander, but they can provide an assessment of possible future actions, and give a level of confidence of those assessments based on the accuracy of the sensor information received.

Another potential real-time application of agent-based simulation is in identifying high-risk areas as they develop. For example, a commander observing his troops in real time could be assisted by a computer that analyzes battlefield patterns and other parameters for the purpose of sounding the alarm when a segment of the battlefield has entered a dynamic that is now heavily information dependent. The commander can now focus on this segment of the battlefield and take actions to ensure that forces in that area are given ample information resources.

4. Example Risk Analysis

To give an idea of how an agent-based simulation can be used to perform a risk analysis, the attrition scenario is explored in greater detail. The standard deviation bars in the graphs generated for this thesis have not received much comment because we have not framed the discussions in terms of risk. Recall that the bars represent one standard deviation from the mean. The large variation represented by these bars in some of the previous simulations does not mean that the scenario must be run hundreds of times before a conclusion can be reached. Scenarios that yield large deviations from the norm (see Figure 5 for an example) suggest that the result can vary wildly with a small change in the initial conditions. In SInBaD, there are two sources of randomness that make one simulation run different from another: the initial placement of forces in their specified cells, and the algorithm used to determine a hit on an opponent (see equation 3.1). Slight differences in the initial placement of agents will alter their paths. This effect combined with the randomness in the hit determination algorithm can lead to large variations in the results. It is this lack of a predictable outcome due to small perturbations in the scenario that defines the scenario as high risk.

The attrition scenario discussed in the validation section of this chapter is a good example. The simulation uses the parameters established in Table 2 and results are reproduced in Figure 20.

The attrition scenario's force position has a large deviation from the average because there were several runs when Blue drove the Red force to the far end of the screen, and there were several runs when Red did the same to Blue. From a planning

point of view, this is a risky scenario and steps could be taken to reduce the risk. Identifying situations of high risk and then determining actions to reduce that risk is where agent-based simulation can be effective.

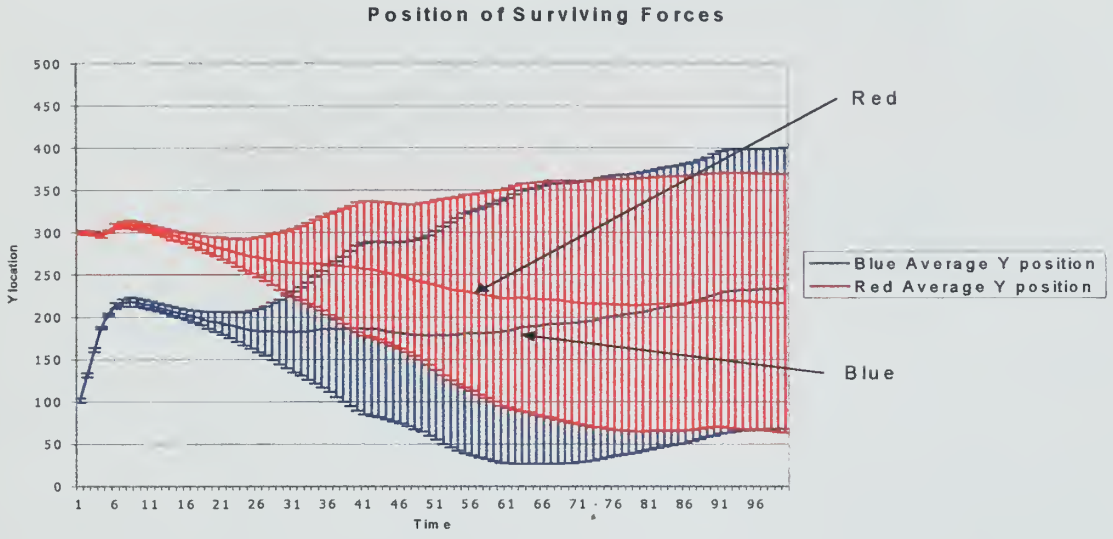

Figure 20 Force Location Attrition Scenario

As an example of how a simulation like SInBaD can be used in this manner, we will look at what can be done to improve Blue's effectiveness in the attrition scenario. Keep in mind that SInBaD is a very abstract simulation. The results of this exercise are not to be considered directly applicable to actual combat. Its purpose is to show how running simulations can provide direction in planning and real time threat assessment. A much more sophisticated simulation program would be necessary to provide meaningful results to a particular combat scenario.

The goal is to find a parameter (preferably one that is measurable in the field) that has a significant impact on the outcome of the battle. The attrition scenario is characterized by static lines of battle and it was observed that changes in information parameters have a smaller impact in these situations. One might suspect that the attrition scenario would be heavily dependent on the number of forces in the field. Just how sensitive this scenario is to this parameter can only be guessed without this type of simulation.

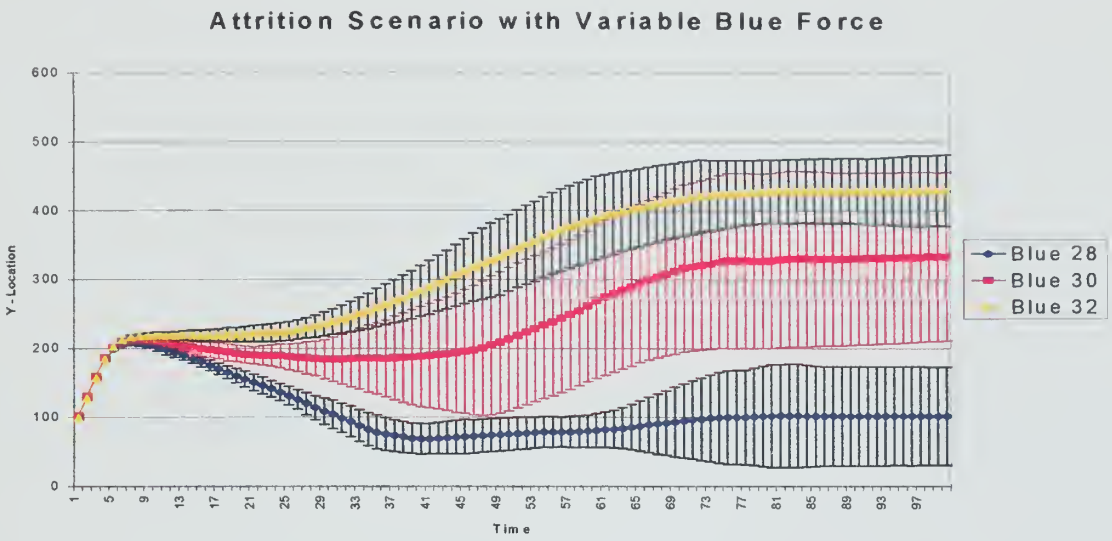

Figure 21 Attrition Scenario with Variable Blue Force

Figure 21 shows the average location of the Blue force when the Blue to Red force ratio is 28 to 30, 30 to 30 (original scenario) and 32 to 30. The results indicate that this scenario is highly sensitive to the initial number of Blue troops. With the ratio at 28 to 30, Blue is consistently driven back. An increase of just 4 agents yields the opposite result. The 32/30 scenario shows a smaller deviation from the mean indicating that the result is more consistent than the 30/30 scenario, it is also a more desirable result. If a more realistic scenario had yielded these results, it would be wise for a planner to ensure that the Blue to Red force ratio stayed above 32 to 30.

V. FUTURE PROJECTS

Major advances in computer technology coupled with extensive research in agent-based modeling are required to make this type of simulation a truly useful decision-making aid. Developing and running simulations like SInBaD and ISAAC generate the insights that allow researchers to take the next step in attaining that goal. During the course of this thesis, several additions to SInBaD were contemplated that would make it a more useful and user friendly experimental tool.

It is important to note that the Marine Corps is still supporting the ISAAC simulation which is now in a follow on implementation called EINSTein (Enhanced ISAAC Neural Simulation Tool). EINSTein continues to maintain the cutting edge for agent-based combat simulation with many of the suggestions below already incorporated. Anyone interested in improving SInBad or creating their own simulation is advised to stay informed on the developments of the EINSTein simulation system.

A. USER INTERFACE

Considering the experience of the programmer and the time restriction inherent in writing a thesis, it is no wonder that the user interface of SInBaD is less than desirable. It is recommended that future versions be completely encapsulated in a Windows environment to include the following features:

1. Incorporate the input, execution, and visual display features into one Windows based GUI environment.
2. Provide pull-down menus and radio buttons to insert input data for simulation runs.
3. Provide the ability to save output results (both visual and statistical) to a specified file name.
4. Provide tighter integration with Microsoft Excel to allow more flexibility in the determination of measurable outputs and the better display of statistical results.
5. Provide the ability to alter agent parameters while the simulation is being viewed.

B. INCREASED EFFICIENCY AND FIDELITY

Running the simulations created more questions than answers. The following improvements go a long way to increasing SInBaD's flexibility and utility.

1. The current version of SInBaD runs very slowly when agent populations reached 100. An experienced programmer should be able to squeeze more efficiency from the original algorithms.
2. The agents are based on a single class called "squad". Separating some of the functionality that is completely encapsulated in the "squad class" would make upgrading easier. An example is the communications capability of the agents. Objectifying this ability would make it easier to simulate specific types of communications capability.
3. Discussions with several top simulation experts have made one point very clear: simulation of a particular scenario is not possible without a terrain model. A first iteration could be the addition of simple barriers and rough spots to determine the theoretical consequences of terrain.
4. Providing a targeting scheduling algorithm to SInBaD agents would be a nice addition and allow for the analysis of the interplay between scheduling algorithms and the level of autonomy of command.

C. ADAPTABILITY

The modeling of any collection of intelligent entities should include the ability for agents to learn from previous experiences. One of the biggest benefits to this type of simulation is that computer simulated agents can learn and develop new survival strategies that are not realizable by any other means. Computer programming techniques like genetic algorithms can search and find "good" solutions in a continually evolving parametric landscape. Tierra, created by Tom Ray and discussed in chapter 2, is a good example of a simulation that provides a mechanism for competing digital organisms to evolve and be rewarded for developing more effective survival strategies.

Providing agent adaptability not only helps in the understanding of current tactics and strategies, but also will aid in the creation of new knowledge in these areas. Some ideas for incorporating adaptation are:

1. Incorporate a Genetic Programming algorithm. A genetic program is similar in concept to a genetic algorithm. Instead of defining a fitness function for an

individual agent in the simulation and then letting agents compete against each other (the genetic algorithm model), a fitness function for the entire simulation is created and individual programs compete. This makes more sense since the goal of combat is not the attainment of an objective by an individual actor but by an entire ensemble of actors.

2. The ISAAC simulation discussed earlier allows the simulator to set different parameters for agents once they are hit once. A second hit removes the agent from play. Although this allows for different behaviors during the simulation, the changes are made explicitly. In keeping with the agent-based paradigm, a mechanism that allows for parametric changes based on internal rules should be developed.
3. Another mechanism for adaptation could be through communication between agents. Agents could pass information (including erroneous information) that would cause their internal rule set to be modified.
4. Agent parameters could be changed as a function of information quality. This would allow for the alteration of agent behavior when it is cut off from its source of information.

APPENDIX. INSTALLATION AND PROGRAM CODE

A. USER'S MANUAL

1. Requirements

SInBaD was developed and run on a 200 MHz Pentium IBM compatible computer running the Microsoft Windows 95 operating system. Any computer capable of operating Windows 95 should be able to run SInBaD. 32 Mb of RAM was used for most of the simulations but 64 Mb is recommended.

2. Installation

SInBaD is not professionally written software and some effort is required to install it. It is very stable and should run without incident once it is installed.

The SInBaD simulation includes the following files:

1. Sinbad.exe – SInBaD executable file*
2. datain.txt – input file*
3. data.txt – output file (created if not included)
4. Sinbad.tk – visual display script
5. data.xls – an Excel file which can be used as a template to develop statistical time plots.*

Copy all 5 files into a directory named **Sinbad**. If you want to view the visual display, you must install TCL 7.5/TK 4.1 or greater. TCL/TK is a free scripting language compiler that drives the visual display and can be obtained at ftp://ftp.scriptics.com/pub/tcl/7_5. Look for a file like **win41.exe**.

Once TCL/TK is installed, go into the **tcl\bin** directory and copy the following files to the directory that has your SInBaD installation;

1. Wish41.exe*
2. Tk41.dll
3. Tcl75.dll
4. CW3215.dll

The simplest way to run SInBaD is to build a **SInBaD folder** on your desktop and place shortcuts to the files marked with a *. These are the files that you will be interacting with. To place a folder on the Windows desktop, find a blank space on the desktop and right click on the mouse. Go to **New** and then **Folder**. Label the folder **SInBaD**.

Once the shortcuts are in place, go to the shortcut called "wish41.exe", right click on the mouse and go to **Properties**. If your **target** line reads **"C:\Sinbad\wish41.exe"**, change it to **"C:\Sinbad\wish41.exe" sinbad.tk**. Adjust your target line as necessary.

3. Running SInBaD

To adjust the input parameters enter the **SInBaD folder** you created, double click on the **datain.txt** file. All user defined parameters for SInBaD agents are located here. SInBaD allows for the independent creation of up to 3 Blue and 3 Red cells. To turn a cell off, simply set the number of agents in that cell to 0. The following defines each cell parameter:

```
20      agents in cell 1
300     cell 1 box center x - initial placement of agents
100     cell 1 box center y - initial placement of agents
300     cell 1 box size x - initial size of cell in x-direction
50      cell 1 box size y - initial size of cell in y-direction
300     cell 1 goal x (Blue cells only)
500     cell 1 goal y (Blue cells only)
.01     cell 1 pk - defines probability of hit for agents in
        that cell
2       cell 1 speed
100     cell 1 sensor/shooter range
.5      cell 1 charge ratio - enemy-to-friendly ratio of agents
        (see chapter 3)
1.5     cell 1 runaway ratio - enemy-to-friendly ratio of
        agents (see chapter 3)
3       cell 1 max hits - number of hits an agent can take
        before being removed from play
```

The following parameters are either not agent specific or control all agents on the side designated in the description.

```
2000    run size (number of time steps) - sets the number of
        steps the simulation will run.
```

```
5       simulation speed (display mode only) - speeds up the
        graphical display by showing only the nth step defined
        by this parameter (has no effect on the agents
        themselves.  When simulation is in batch mode,
        parameter is disabled.
80      Blue affinity distance (must be less than view
        distance) - see chapter 3 for description
80      Red affinity distance (must be less than view distance)
        - see chapter 3 for description
0       number of steps before Red receives an info update -
        see chapter 3 for description
0       number of steps before Blue receives an info update -
        see chapter 3 for description
0       position reporting error for Red  - see chapter 3 for
        description
0       position reporting error for Blue - see chapter 3 for
        description
8       number of Blue around before attacking (less than
        bounded rationality) - see chapter 3 for description
0       number of Red around before attacking (less than
        bounded rationality) - see chapter 3 for description
0       flag for display(0) or batch mode(1) - a "0" generates
        an output file (data.txt) that is readable by the
        sinbad.tk script and will display visual results when
        "wish41.exe" is run.  When set to "1", an output file
        is generated that is formatted to the Excel template
        provided.
10      number of runs (batch mode) - sets the number of runs
        that will be averaged to produce the output file used
        in statistical analysis (disabled when display flag is
        set to 0).
```

To run the parameter set you created, save your changes to **datain.txt** and run **sinbad.exe** by double clicking the **sinbad.exe** shortcut in your SInBaD folder. The output file **data.txt** will be generated, or written over if one is already present. If the "display flag" was set, double click on the shortcut pointing to **"wish41.exe" sinbad.tk** to view the results.

If the **batch flag** was set, then import the **data.txt** into the **data.xls** Excel file. *Note: It is recommended that a macro be created to automatically perform this task for you.*

B. PROGRAM CODE

The following is all the program code used to develop the SInBaD simulation system.

1. Input File

```
20          Blue agents in cell 1
300         cell 1 box center x
100         cell 1 box center y
300         cell 1 box size x
50          cell 1 box size y
300         cell 1 goal x
500         cell 1 goal y
.01         cell 1 pk
2           cell 1 speed
100         cell 1 sensor/shooter range
.5          cell 1 charge ratio
1.5         cell 1 runaway ratio
3           cell 1 max hits

0           Blue agents in cell 2
150         cell 2 box center x
100         cell 2 box center y
100         cell 2 box size x
50          cell 2 box size y
300         cell 2 goal x
500         cell 2 goal y
.01         cell 2 pk
2           cell 2 speed
100         cell 2 sensor/shooter range
.5          cell 2 charge ratio
1           cell 2 runaway ratio
3           cell 2 max hits

0           Blue agents in cell 3
500         cell 3 box center x
100         cell 3 box center y
10          cell 3 box size x
10          cell 3 box size y
500         cell 3 goal x
500         cell 3 goal y
.01         cell 3 pk
2           cell 3 speed
75          cell 3 sensor/shooter range
.5          cell 3 charge ratio
1           cell 3 runaway ratio
3           cell 3 max hits

20          Red agents in cell 1
300         cell 1 box center x
300         cell 1 box center y
200         cell 1 box size x
50          cell 1 box size y
.01         cell 1 pk
```

```
2           cell 1 speed
100         cell 1 sensor/shooter range
.5          cell 1 charge ratio
1           cell 1 runaway ratio
3           cell 1 max hits

0           Red agents in cell 2
300         cell 2 box center x
300         cell 2 box center y
300         cell 2 box size x
50          cell 2 box size y
.5          cell 2 pk
2           cell 2 speed
100         cell 2 sensor/shooter range
.5          cell 2 charge ratio
1           cell 2 runaway ratio
3           cell 2 max hits

0           Red agents in cell 3
300         cell 3 box center x
400         cell 3 box center y
100         cell 3 box size x
30          cell 3 box size y
.5          cell 3 pk
2           cell 3 speed
100         cell 3 sensor/shooter range
.5          cell 3 charge ratio
1           cell 3 runaway ratio
10          cell 3 max hits

2000        run size (number of time steps)
5           simulation speed (display mode only)
80          Blue affinity distance (must be less than view distance)
80          Red affinity distance (must be less than view distance)
0           number of steps before Red receives an info update
0           number of steps before Blue receives an info update
0           position reporting error for Red
0           position reporting error for Blue
8           number of Blue around before attacking (less than bounded
            rationality)
0           number of Red around before attacking (less than bounded
            rationality)
0           flag for display(0) or batch mode(1)
10          number of runs (batch mode)
```

2. Sinbad.tk script

```
# Program reads file data.txt one line at a time and
#     displays it on the screen.


proc input {fileid} {
      gets $fileid data
      return $data
}


canvas .can \
      -width 1000  \
      -height 1000 \
      -background white
pack .can
update
# after 50

set filename "data.txt"
if {[file readable $filename]} {
  set fileid [open $filename "r"]
  set data [input $fileid]
  scan "$data" "%d %d %d" step Blue Red}
  set totalObjects [expr $Blue + $Red]

  for {set i 0} {$i < $totalObjects} {incr i} {
    set data [input $fileid]
    scan "$data" "%d %f %f" col posx($i) posy($i)
    set color Red
    if {$col == 1} {set color Blue}

    set tag($i) [.can create oval $posx($i) $posy($i) \
            [expr $posx($i)+10] [expr $posy($i)+10] -fill $color]
    update
#    after 10
  }

  for {set i 1} {$i < $step} {incr i} {
    for {set j 0} {$j < $totalObjects} {incr j} {
      set data [input $fileid]
      scan "$data" "%d %f %f" col posx($i) posy($i)
      .can coords $tag($j) $posx($i) $posy($i) \
            [expr $posx($i)+10] [expr $posy($i)+10]
      update
#      after 5
    }
  }

close $fileid
```

3. SInBaD source code

Combat.cpp (main file)

```
//  Richard Hencke - Thesis Project
//
//  Purpose:  Combat sim to model the effects of information and
//    command and control.
//
//  Introduction:  It is the intent of this program to model the effects
//          of information parameters on combat under differing command
structures.
//          This will be accomplished by using an object oriented
//          approach.  A class will represent a single entity of combat power.
//          It is hoped through theThe initial model will consist of opposing
teams.
//
//
//                Later additions will include higher fidelity, adaptive
agents, and
//          and an integrated GUI interface.
//
//  Start date:  November 10, 1997
//  Version 1.0  Complete July 25, 1998

#include<iostream.h>
#include<iomanip.h>
#include<stdlib.h>
#include<conio.h>
#include<fstream>
#include<stdlib.h>
#include<math.h>
#include "squad.h"
#include "comm.h"



int main()
{
   enum flag {OFF, ON};
   enum Shooter {NO, YES};
   enum state {DEAD, HIDING = 0, ALIVE = 1};
   enum Runflag {DISPLAY = 0, BATCH = 1};
   Runflag runFlag;
   flag lethality = OFF;
      int redAggressive, blueAggressive;
      int runNumber;

      void output(int, float, float, ofstream &);
   int finalBluex1, finalBluey1, finalBluex2, finalBluey2,
                        finalBluex3, finalBluey3;
   int blueNumber, redNumber;
   int redCell1, redCell2, redCell3, blueCell1, blueCell2, blueCell3;
```

```
int randBlue;
float blueSensorRange1, blueSensorRange2, blueSensorRange3,
                redSensorRange1, redSensorRange2, redSensorRange3;
float blueRisk, redRisk;
int RUNSIZE;
int redBoxCentery1, redBoxSizey1, redBoxCentery2, redBoxSizey2,
                      redBoxCentery3, redBoxSizey3;
int redBoxCenterx1, redBoxSizex1, redBoxCenterx2, redBoxSizex2,
                      redBoxCenterx3, redBoxSizex3;
int blueBoxCenterx1, blueBoxSizex1, blueBoxCenterx2, blueBoxSizex2,
                      blueBoxCenterx3, blueBoxSizex3;
int blueBoxCentery1, blueBoxSizey1, blueBoxCentery2, blueBoxSizey2,
                      blueBoxCentery3, blueBoxSizey3;
int simSpeed;
int redUpdate, blueUpdate;
int runFlagInput;
float blueSpeed1, blueSpeed2, blueSpeed3, redSpeed1, redSpeed2, redSpeed3;
float bluePk1, bluePk2, bluePk3, redPk1, redPk2, redPk3;
float blueCharge1, blueCharge2, blueCharge3,
                redCharge1, redCharge2, redCharge3;
float blueRunaway1, blueRunaway2, blueRunaway3,
                redRunaway1, redRunaway2, redRunaway3;
int blueMaxHits1, blueMaxHits2, blueMaxHits3, redMaxHits1, redMaxHits2,
         redMaxHits3;
int blueError, redError;


   //  Simulation parameters are read in from an input file

const int BUFFER = 150;
char dummyBuffer[BUFFER];
   ifstream inputFile("datain.txt", ios::in);


if(!inputFile)
{
   cerr << "File could not be opened." << endl;
   exit(1);
}

inputFile >> blueCell1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCenterx1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCentery1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizex1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizey1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluex1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluey1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> bluePk1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSpeed1;
```

```
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSensorRange1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueCharge1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueRunaway1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueMaxHits1;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> blueCell2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCenterx2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCentery2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizex2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizey2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluex2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluey2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> bluePk2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSpeed2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSensorRange2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueCharge2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueRunaway2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueMaxHits2;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> blueCell3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCenterx3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxCentery3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizex3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueBoxSizey3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluex3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> finalBluey3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> bluePk3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSpeed3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueSensorRange3;
inputFile.getline(dummyBuffer, BUFFER);
```

```
inputFile >> blueCharge3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueRunaway3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueMaxHits3;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> redCell1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxCenterx1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxCentery1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizex1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizey1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redPk1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSpeed1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSensorRange1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redCharge1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redRunaway1;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redMaxHits1;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> redCell2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxCenterx2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxCentery2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizex2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizey2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redPk2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSpeed2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSensorRange2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redCharge2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redRunaway2;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redMaxHits2;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> redCell3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxCenterx3;
inputFile.getline(dummyBuffer, BUFFER);
```

```
inputFile >> redBoxCentery3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizex3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redBoxSizey3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redPk3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSpeed3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redSensorRange3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redCharge3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redRunaway3;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redMaxHits3;
inputFile.getline(dummyBuffer, BUFFER);

inputFile >> RUNSIZE;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> simSpeed;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueRisk;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redRisk;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redUpdate;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueUpdate;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redError;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueError;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> blueAggressive;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> redAggressive;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> runFlagInput;
inputFile.getline(dummyBuffer, BUFFER);
inputFile >> runNumber;

inputFile.close();

const int ARRAY = 100;
const int REDARRAY = ARRAY;            //  Number of Red Squads
const int BLUEARRAY = ARRAY;           //  Number of Blue Squads
const int BATCHARRAY = 2000;           //  Maximum number of time steps
                                       //          in simulation
BlueSquad Blue[BLUEARRAY];
RedSquad Red[REDARRAY];

        // initialized statistical parameters for all runs

float blueAvgXFinal[BATCHARRAY] = {0};
float blueAvgYFinal[BATCHARRAY] = {0};
```

```
   float redAvgXFinal[BATCHARRAY] = {0};
   float redAvgYFinal[BATCHARRAY] = {0};
   float blueDeadFinal[BATCHARRAY] = {0};
   float redDeadFinal[BATCHARRAY] = {0};
   float blueVarY[BATCHARRAY] = {0};
   float redVarY[BATCHARRAY] = {0};
   float blue2ndMomentY[BATCHARRAY] = {0};
   float red2ndMomentY[BATCHARRAY] = {0};
   float blueDead2ndMoment[BATCHARRAY] = {0};
   float redDead2ndMoment[BATCHARRAY] = {0};
   float blueDeadVar[BATCHARRAY] = {0};
   float redDeadVar[BATCHARRAY] = {0};

            //  initialize pointers to Squads

      Squad *ptrRed[REDARRAY];
      Squad *ptrBlue[BLUEARRAY];

   blueNumber = blueCell1 + blueCell2 + blueCell3;

   if(runFlagInput == 0)            //  flag to determine batch or video mode
   {
      runFlag = DISPLAY;
      runNumber = 1;
   }
   else
   {
      runFlag = BATCH;
      simSpeed = RUNSIZE/100;
   }

   if(redError < 1)
      redError = 1;
   if(blueError < 1)
      blueError = 1;

   if(redUpdate < 1)
      redUpdate = 1;
   if(blueUpdate < 1)
      blueUpdate = 1;

      //  Simulation results are outputted to a file

      ofstream outputFile("data.txt", ios::out);
   if (!outputFile)
   {
      cerr << "File could not be opened" << endl;
      exit(1);
   }

   cout << "   Please wait while Blue and Red duke it out... " << endl;

for(int i = 0; i < runNumber; i++)
{
      srand(time(NULL));
            //  statistical parameters for each run
   float blueAvgX[BATCHARRAY] = {0};
```

```
float blueAvgY[BATCHARRAY] = {0};
float redAvgX[BATCHARRAY] = {0};
float redAvgY[BATCHARRAY] = {0};
float blueDead[BATCHARRAY] = {0};
float redDead[BATCHARRAY] = {0};

cout << "            Battle number " << i+1 << endl;

   // initializing Blue agent parameters

for(int i = 0; i < blueCell1; i++)
   {
   Blue[i].goalx = finalBluex1;
   Blue[i].goaly = finalBluey1;
   ptrBlue[i] = &Blue[i];
   Blue[i].viewDistance = blueSensorRange1;
   Blue[i].id = i + 1;
   Blue[i].speed = blueSpeed1;
   Blue[i].riskDistance = blueRisk;
   Blue[i].agentState = ALIVE;
   Blue[i].pk = bluePk1;
   Blue[i].aggressive = blueAggressive;
   Blue[i].initialPosition(blueBoxCenterx1, blueBoxCentery1,
                                            blueBoxSizex1, blueBoxSizey1);
   Blue[i].charge = blueCharge1;
   Blue[i].runaway = blueRunaway1;
   Blue[i].maxHits = blueMaxHits1;
   Blue[i].hitsTaken = 0;
   Blue[i].hitsGiven = 0;
   Blue[i].targeted = NO;
   Blue[i].shooter = NO;
   Blue[i].update = blueUpdate;
   Blue[i].error = blueError;
}

for(int i = blueCell1; i < (blueCell1 + blueCell2); i++)
   {
   Blue[i].goalx = finalBluex2;
   Blue[i].goaly = finalBluey2;
   ptrBlue[i] = &Blue[i];
   Blue[i].viewDistance = blueSensorRange2;
   Blue[i].id = i + 1;
   Blue[i].speed = blueSpeed2;
   Blue[i].riskDistance = blueRisk;
   Blue[i].agentState = ALIVE;
   Blue[i].pk = bluePk2;
   Blue[i].aggressive = blueAggressive;
   Blue[i].initialPosition(blueBoxCenterx2, blueBoxCentery2,
                                            blueBoxSizex2, blueBoxSizey2);
   Blue[i].charge = blueCharge2;
   Blue[i].runaway = blueRunaway2;
   Blue[i].maxHits = blueMaxHits2;
   Blue[i].hitsTaken = 0;
   Blue[i].hitsGiven = 0;
   Blue[i].targeted = NO;
   Blue[i].shooter = NO;
   Blue[i].update = blueUpdate;
```

```
   Blue[i].error = blueError;
}

for(int i = (blueCell1 + blueCell2); i <
                                (blueCell1 + blueCell2 + blueCell3); i++)
   {
   Blue[i].goalx = finalBluex3;
   Blue[i].goaly = finalBluey3;
   ptrBlue[i] = &Blue[i];
   Blue[i].viewDistance = blueSensorRange3;
   Blue[i].id = i + 1;
   Blue[i].speed = blueSpeed3;
   Blue[i].riskDistance = blueRisk;
   Blue[i].agentState = ALIVE;
   Blue[i].pk = bluePk3;
   Blue[i].aggressive = blueAggressive;
   Blue[i].initialPosition(blueBoxCenterx3, blueBoxCentery3,
                                          blueBoxSizex3, blueBoxSizey3);
   Blue[i].charge = blueCharge3;
   Blue[i].runaway = blueRunaway3;
   Blue[i].maxHits = blueMaxHits3;
   Blue[i].hitsTaken = 0;
   Blue[i].hitsGiven = 0;
   Blue[i].targeted = NO;
   Blue[i].shooter = NO;
   Blue[i].update = blueUpdate;
   Blue[i].error = blueError;
}

redNumber = redCell1 + redCell2 + redCell3;

         // initialize parameters for Red agents

   for(int i = 0; i < redCell1; i++)
{
   ptrRed[i] = &Red[i];
   Red[i].id = i + 1;
   Red[i].speed = redSpeed1;
   Red[i].viewDistance = redSensorRange1;
   Red[i].riskDistance = redRisk;
   Red[i].update = redUpdate;
   Red[i].agentState = ALIVE;
   Red[i].pk = redPk1;
   Red[i].aggressive = redAggressive;
   Red[i].initialPosition(redBoxCenterx1, redBoxCentery1,
                                          redBoxSizex1, redBoxSizey1);
   Red[i].charge = redCharge1;
   Red[i].runaway = redRunaway1;
   Red[i].maxHits = redMaxHits1;
   Red[i].hitsTaken = 0;
   Red[i].hitsGiven = 0;
   Red[i].targeted = NO;
   Red[i].shooter = NO;
   Red[i].error = redError;
}
```

```
   for(int i = (redCell1); i < (redCell1 + redCell2); i++)
{
   ptrRed[i] = &Red[i];
   Red[i].id = i + 1;
   Red[i].speed = redSpeed2;
   Red[i].viewDistance = redSensorRange2;
   Red[i].riskDistance = redRisk;
   Red[i].update = redUpdate;
   Red[i].agentState = ALIVE;
   Red[i].pk = redPk2;
   Red[i].aggressive = redAggressive;
   Red[i].initialPosition(redBoxCenterx2, redBoxCentery2,
                                          redBoxSizex2, redBoxSizey2);
   Red[i].charge = redCharge2;
   Red[i].runaway = redRunaway2;
   Red[i].maxHits = redMaxHits2;
   Red[i].hitsTaken = 0;
   Red[i].hitsGiven = 0;
   Red[i].targeted = NO;
   Red[i].shooter = NO;
   Red[i].error = redError;
}


   for(int i = (redCell1 + redCell2 ); i <
                                (redCell1 + redCell2 + redCell3); i++)
{
   ptrRed[i] = &Red[i];
   Red[i].id = i + 1;
   Red[i].speed = redSpeed3;
   Red[i].viewDistance = redSensorRange3;
   Red[i].riskDistance = redRisk;
   Red[i].update = redUpdate;
   Red[i].agentState = ALIVE;
   Red[i].pk = redPk3;
   Red[i].aggressive = redAggressive;
   Red[i].initialPosition(redBoxCenterx3, redBoxCentery3,
                                redBoxSizex3, redBoxSizey3);
   Red[i].charge = redCharge3;
   Red[i].runaway = redRunaway3;
   Red[i].maxHits = redMaxHits3;
   Red[i].hitsTaken = 0;
   Red[i].hitsGiven = 0;
   Red[i].targeted = NO;
   Red[i].shooter = NO;
   Red[i].error = redError;
}


         // Output program parameters needed to run the graphics simulation

   if(runFlag == DISPLAY)
{
         outputFile << RUNSIZE/simSpeed << " " << blueNumber << " "
                    << redNumber << endl;
   for (int timeStep = 0; timeStep < RUNSIZE; timeStep++)
      {
```

```
          for(int i = 0; i < blueNumber; i++)
            {
            Blue[i].timeStep = timeStep;
                  Blue[i].blueDecide(ptrBlue, ptrRed, blueNumber, redNumber);

                  if(timeStep%simSpeed == 0)
                        output(1, Blue[i].position.x, Blue[i].position.y,
outputFile);
            }

         for(int i = 0; i < redNumber; i++)
            {
            Red[i].timeStep = timeStep;
                  Red[i].redDecide(ptrBlue, ptrRed, blueNumber, redNumber);

                  if(timeStep%simSpeed == 0)
                        output(2, Red[i].position.x, Red[i].position.y,
outputFile);
            }
      }
   }
   else
   {
      for (int timeStep = 0; timeStep < RUNSIZE; timeStep++)
         {
         float sumx = 0;
         float sumy = 0;

                  for(int i = 0; i < blueNumber; i++)
            {
            Blue[i].timeStep = timeStep;
                  Blue[i].blueDecide(ptrBlue, ptrRed, blueNumber, redNumber);
            sumx = sumx + Blue[i].position.x;
            sumy = sumy + Blue[i].position.y;
            if(Blue[i].agentState == DEAD)
                  blueDead[timeStep] = blueDead[timeStep] + 1;
            }
         blueAvgX[timeStep] = sumx/blueNumber;
         blueAvgY[timeStep] = sumy/blueNumber;

         sumx = 0;
         sumy = 0;

         for(int i = 0; i < redNumber; i++)
            {
            Red[i].timeStep = timeStep;
            Red[i].redDecide(ptrBlue, ptrRed, blueNumber, redNumber);
            sumx = sumx + Red[i].position.x;
            sumy = sumy + Red[i].position.y;

            if(Red[i].agentState == DEAD)
                  redDead[timeStep] = redDead[timeStep] + 1;
            }
         redAvgX[timeStep] = sumx/redNumber;
         redAvgY[timeStep] = sumy/redNumber;
      }
   }
```

```
      for(int i = 0; i < RUNSIZE; i++)
   {
      blueAvgXFinal[i] = blueAvgXFinal[i] + blueAvgX[i];
      blueAvgYFinal[i] = blueAvgYFinal[i] + blueAvgY[i];
      blueDeadFinal[i] = blueDeadFinal[i] + blueDead[i];
      redAvgXFinal[i] = redAvgXFinal[i] + redAvgX[i];
      redAvgYFinal[i] = redAvgYFinal[i] + redAvgY[i];
      redDeadFinal[i] = redDeadFinal[i] + redDead[i];

      blue2ndMomentY[i] = blue2ndMomentY[i] + blueAvgY[i]*blueAvgY[i];
      red2ndMomentY[i] = red2ndMomentY[i] + redAvgY[i]*redAvgY[i];
      blueDead2ndMoment[i] = blueDead2ndMoment[i] + blueDead[i]*blueDead[i];
      redDead2ndMoment[i] = redDead2ndMoment[i] + redDead[i]*redDead[i];
   }
}
      if(runFlag == BATCH)
   {
         for(int i = 0; i < RUNSIZE; i++)
      {
            blueAvgXFinal[i] = blueAvgXFinal[i]/runNumber;
            blueAvgYFinal[i] = blueAvgYFinal[i]/runNumber;
            blueDeadFinal[i] = blueDeadFinal[i]/runNumber;
            redAvgXFinal[i] = redAvgXFinal[i]/runNumber;
            redAvgYFinal[i] = redAvgYFinal[i]/runNumber;
            redDeadFinal[i] = redDeadFinal[i]/runNumber;
            blueVarY[i] = pow((blue2ndMomentY[i]/runNumber -
                              blueAvgYFinal[i]*blueAvgYFinal[i]),.5);
            redVarY[i] = pow((red2ndMomentY[i]/runNumber -
                              redAvgYFinal[i]*redAvgYFinal[i]),.5);
            redDeadVar[i] = pow((redDead2ndMoment[i]/runNumber -
                              redDeadFinal[i]*redDeadFinal[i]),.5);
            blueDeadVar[i] = pow((blueDead2ndMoment[i]/runNumber -
                              blueDeadFinal[i]*blueDeadFinal[i]),.5);
       }

      outputFile << "  Blue Avg. X  Blue Avg. Y  Red Avg. X  Red Avg. Y"
                 << "  Blue Dead  Red Dead  Blue Var   Red Var  "
                 << "  Blue Dead Var   Red Dead Var"
                 << endl << endl;
            for(int i = 0; i < RUNSIZE; i++)
      {
                  if(i%simSpeed == 0)
                        outputFile << setw(10)
                        << blueAvgXFinal[i] << setw(13)
                        << blueAvgYFinal[i] << setw(13)
                        << redAvgXFinal[i]  << setw(13)
                        << redAvgYFinal[i]  << setw(9)  << setprecision(4)
                        << blueDeadFinal[i] << setw(9)  << setprecision(4)
                        << redDeadFinal[i]  << setw(13) << setprecision(4)
                        << blueVarY[i]         << setw(13) << setprecision(4)
                        << redVarY[i]          << setw(13) << setprecision(4)
                        << blueDeadVar[i]   << setw(13) << setprecision(4)
                        << redDeadVar[i]    << setw(13)
                        << endl;
            }
   }
   outputFile.close();
```

```
	return 0;
}

	//  Output position to file

void output(int color, float posx, float posy, ofstream &outputFile)
{
	outputFile  << color << " " << posx << " " << posy    << endl;
}
```

Squad.h

```
#ifndef squad_h
#define squad_h

#include<iostream>
#include<time.h>
#include<stdlib.h>
#include<stddef.h>


//  Base Squad from which Blue and Red Squads are derived

const int pi = 3.14159265359;
class Squad
{
public:
   struct Position                          // Tracks current location of agents
   {
      float x;
      float y;
   };

   struct List          //  List tracks enemy and friendly agents each
   {                    //          each agent sees.
      int id;
      float positionx;
      float positiony;
      float direction;
      float distance;
   };

   List redList[20];          //  Holds info on agents in viewing area
   List blueList[20];

   List redRisk[20];          //  Holds info on agents in risk area
   List blueRisk[20];

   int boundedRationality;

   Position position;              //  Identifies position of squad
   Position positionComm;          //  Holds value of position with error

   enum State {DEAD = 0, HIDING = 0, ALIVE = 1};
   enum Shooter {NO, YES};
      State agentState;
   Shooter shooter, aggressive, targeted;

   int id;
   int goalx, goaly;             //  Currently established long term goal
   float fixx, fixy;             //  Currently established short term goal
   float viewDistance;                    // Distance squad can see
   float riskDistance;                    // Level of risk aversity
   float direction;
```

```
   int lethality;
   int redViewCount, blueViewCount;
   int redRiskCount, blueRiskCount;
   int timeStep;
   float speed;
   float pk;                          // Agent probability of hit
   int detection;              // Flag notifies agent he has been detected
   int update;                 // Parameter denotes rapidness of info updates;
   int boxcenterx, boxcentery;
   int boxsizex, boxsizey;
   float runaway, charge;                  // Enemy-to-force ratio parameters
   int hitsTaken, hitsGiven, maxHits;
   int error;                  // Info parameter used to induce info error

public:
      Squad()
   {
      position.x=0;
      position.y=0;
      detection = 0;
      targeted = NO;
      shooter = NO;
      hitsTaken = 0;
      hitsGiven = 0;
      boundedRationality = 10;
   };

      Squad(float xstart, float ystart, int)
   {
      detection = 0;
      position.x= xstart;
      position.y= ystart;
      targeted = NO;
      shooter = NO;
      hitsTaken = 0;
      hitsGiven = 0;
      boundedRationality = 10;
   }

   void move(float, float);  // Move to position identified by inputs
      int randFunction(int);    // Produces a random function with modulo
                                //   of input
   void initialPosition(int, int, int, int);
};


// Blue Squad class definition

class BlueSquad : public Squad
{
public:


      BlueSquad()
   {
      position.x = 0;
      position.y = 0;
```

```
    }
    void blueDecide(Squad * *, Squad * *, const int, const int);
    void infoProcess();
    void surveyArea(Squad * *, Squad * *, const int, const int);
    void riskArea(Squad * *, Squad * *, const int, const int);
    void shootDecision(Squad * *, Squad * *, const int , const int);

};     // end BlueSquad


//  Red Squad class

class RedSquad : public Squad
{
public:
    int upperRightx, upperRighty;
    int upperLeftx, upperLefty;
    int lowerRightx, lowerRighty;
    int lowerLeftx, lowerLefty;

       RedSquad()
    {
       position.x = 0;                                    (
       position.y = 0;
    }

    void redDecide(Squad * *, Squad * *, const int , const int );
    void buildArea(int, int, int, int);
    void infoProcess();
    void surveyArea(Squad * *, Squad * *, const int, const int);
    void riskArea(Squad * *, Squad * *, const int, const int);
    void shootDecision(Squad * *, Squad * *, const int , const int);

};     //  end RedSquad

#endif
```

Squad.cpp

```
#include "squad.h"
#include<conio.h>
#include <math.h>


//*************************************************************************
//                        Red Squad functions
//*************************************************************************

//  Decide function is each agents OODA loop

void RedSquad::redDecide(Squad * *blueActor, Squad * *redActor,
                                          const int blueTotal, const int
redTotal)
{
   if(agentState == ALIVE)
      {
      // First step in OODA loop (Observe)
            surveyArea(blueActor, redActor, blueTotal, redTotal);
            riskArea(blueActor, redActor, blueTotal, redTotal);

      // Second and Third step in OODA loop (Orient and Decide)
            if(timeStep%update == 0)
                  infoProcess();

      // Fourth step in OODA loop (Act)
      shootDecision(blueActor, redActor, blueTotal, redTotal);
            move(fixx, fixy);
   }
}
   // Survey area has Red agent looking around him in a circle of
   //   size viewDistance.
void RedSquad::surveyArea(Squad * *blueActor, Squad * *redActor,
                                          const int blueTotal, const int
redTotal)
{
   redViewCount = 0;
   blueViewCount = 0;

   //  Loop scans for Blue forces in area defined by ViewDistance, places all
      //    that meet criteria in a list
      for(int i = 0; i < blueTotal; i++)
   {
      float distance = viewDistance;
if((distance = (sqrt(pow((blueActor[i]->position.x - position.x),2) +
            pow((blueActor[i]->position.y - position.y),2)))) < viewDistance
            && blueActor[i]->agentState == ALIVE)
      {
            detection++;
            if (blueViewCount < boundedRationality)
         {
                  blueList[blueViewCount].id = blueActor[i]->id;
            blueList[blueViewCount].positionx = blueActor[i]->position.x+
```

```
                        randFunction(2*error) - error;
                blueList[blueViewCount].positiony = blueActor[i]->position.y+
                        randFunction(2*error) - error;
                blueList[blueViewCount].direction = blueActor[i]->direction;
                blueList[blueViewCount].distance = distance;
                blueViewCount++;
            }
        }
        }

        //  Scans for Red forces in area defined by viewDistance, places all
        //    that meet criteria in a list
        for(int i = 0; i < redTotal; i++)
    {
        float distance = viewDistance;
        if((distance = (sqrt(pow((redActor[i]->position.x - position.x),2) +
                pow((redActor[i]->position.y - position.y),2)))) < viewDistance
                && redActor[i]->agentState == ALIVE)
        {
                if (redViewCount < boundedRationality)
            {
                redList[redViewCount].id = redActor[i]->id;
                redList[redViewCount].positionx = redActor[i]->position.x+
                        randFunction(2*error) - error;
                redList[redViewCount].positiony = redActor[i]->position.y+
                        randFunction(2*error) - error;
                redList[redViewCount].direction = redActor[i]->direction;
                redList[redViewCount].distance = distance;
                redViewCount++;
            }
        }
    }
}

    // Risk area has Red agent looking around him in a circle of
    //   size by riskDistance.  Risk area defines the range used in
    //   the affinity parameter.
void RedSquad::riskArea(Squad * *blueActor, Squad * *redActor,
                                    const int blueTotal, const int redTotal)
{
    redRiskCount = 0;
    blueRiskCount = 0;

        //  scans for Blue forces in area defined by riskDistance, places all
        //    that meet criteria in a list
        for(int i = 0; i < blueTotal; i++)
    {
        if(sqrt(pow((blueActor[i]->position.x - position.x),2) +
                pow((blueActor[i]->position.y - position.y),2)) < riskDistance
                && blueActor[i]->agentState == ALIVE)
        {
                detection++;
                if (blueRiskCount < boundedRationality)
            {
                blueRisk[blueRiskCount].id = blueActor[i]->id;
                blueRisk[blueRiskCount].positionx = blueActor[i]->position.x+
                        randFunction(2*error) - error;
```

```
            blueRisk[blueRiskCount].positiony = blueActor[i]->position.y+
                        randFunction(2*error) - error;
            blueRisk[blueRiskCount].direction = blueActor[i]->direction;
            blueRiskCount++;
         }
      }
      }

      //  scans for Red forces in area defined by riskDistance, places all
      //    that meet criteria in a list
      for(int i = 0; i < redTotal; i++)
   {
      if(sqrt(pow((redActor[i]->position.x - position.x),2) +
            pow((redActor[i]->position.y - position.y),2)) < riskDistance
            && redActor[i]->agentState == ALIVE)
      {
            if (redRiskCount < boundedRationality)
         {
            redRisk[redRiskCount].id = redActor[i]->id;
            redRisk[redRiskCount].positionx = redActor[i]->position.x+
                        randFunction(2*error) - error;
            redRisk[redRiskCount].positiony = redActor[i]->position.y+
                        randFunction(2*error) - error;
            redRisk[redRiskCount].direction = redActor[i]->direction;
            redRiskCount++;
         }
      }
   }
}

   //       InfoProcess takes the data obtained from surveyArea and Risk Area
   //  and determines average position of opposing forces.  Based on force
   //  projections, it decides whether to run, attack, stay put, or continue
   //  patrol.

void RedSquad::infoProcess()
{
   float averagex = 0;
   float averagey = 0;
   float sumx = 0;
   float sumy = 0;

   if(redRiskCount < aggressive && redViewCount > 0 && blueViewCount > 0)
   {
      for(int i = 0; i < redViewCount; i++)
      {
         sumx = sumx + redList[i].positionx;
         sumy = sumy + redList[i].positiony;
      }
      averagex = sumx/(float)redViewCount;
      averagey = sumy/(float)redViewCount;

      fixx = averagex;
      fixy = averagey;
   }
   else if(blueViewCount == 0 && targeted == NO)
   {
```

```
            if(fabs(fixy - position.y) <= 5 && fabs(fixx - position.x) <= 5)
            {
                  fixx = boxcenterx + randFunction(boxsizex) - boxsizex/2;
                  fixy = boxcentery + randFunction(boxsizey) - boxsizey/2;
      }
   }
   else if(blueViewCount == 0 && targeted == YES)
   {
      for(int i = 0; i < redViewCount; i++)
      {
         sumx = sumx + redList[i].positionx;
         sumy = sumy + redList[i].positiony;
      }
      averagex = sumx/(float)redViewCount;
      averagey = sumy/(float)redViewCount;

      fixx = averagex;
      fixy = averagey;
   }
   else if(((float)blueViewCount/redViewCount) > charge &&
                              ((float)blueViewCount/redViewCount) < runaway)
   {
      fixx = position.x + randFunction(10) - 5;
      fixy = position.y + randFunction(10) - 5;
      }
   else
   {
      for(int i = 0; i < blueViewCount; i++)
         {
         sumx = sumx + blueList[i].positionx;
         sumy = sumy + blueList[i].positiony;
         }
      averagex = sumx/(float)blueViewCount;
      averagey = sumy/(float)blueViewCount;
         if(((float)blueViewCount/redViewCount) <= charge)
      {
            fixx = averagex;
            fixy = averagey;
      }
      else
      {
            fixx = 2*position.x - averagex;
            fixy = 2*position.y - averagey;
      }
   }

}  // end infoProcess


   // Provides the corner points of Red's initial patrol area,
   //  function not currently used.

void RedSquad::buildArea(int centerx, int centery, int sizex, int sizey)
{
   upperRightx = centerx - (float) sizex / 2;
   upperRighty = centery - (float) sizey / 2;
```

```
   upperLeftx  = centerx + (float) sizex / 2;
   upperLefty  = centery - (float) sizey / 2;

   lowerRightx = centerx - (float) sizex / 2;
   lowerRighty = centery + (float) sizey / 2;

   lowerLeftx  = centerx + (float) sizex / 2;
   lowerLefty  = centery + (float) sizey / 2;
}

      //  shootDecision determines if the criteria is met to shoot.

void RedSquad::shootDecision(Squad * *blueActor, Squad * *redActor,
                                    const int blueTotal, const int redTotal)
{

      if(aggressive == NO)
   {
      if(shooter == NO && (targeted == YES || blueRiskCount != 0))
            shooter = YES;
   }
   else
   {
      if(shooter == NO && (targeted == YES || blueViewCount != 0))
                  shooter = YES;
   }

      if(shooter == YES && blueViewCount != 0 && pk != 0)
   {
      for(int i = 0; i < blueViewCount; i++)
         {
            blueActor[blueList[i].id - 1]->targeted = YES;
         float result;
         result = pk*(1 - (pow(blueList[i].distance,2)/pow(viewDistance,2)));
         if((result/blueViewCount) > ((float)rand()/(float)RAND_MAX))
         {
            hitsGiven = hitsGiven + 1;
            blueActor[blueList[i].id - 1]->hitsTaken =
                  blueActor[blueList[i].id - 1]->hitsTaken + 1;
         }
         if(blueActor[blueList[i].id - 1]->hitsTaken >=
                  blueActor[blueList[i].id - 1]->maxHits)
                  blueActor[blueList[i].id - 1]->agentState = DEAD;
      }
      }
   return;
}  // end shootDecision




//*************************************************************************
//                      Blue Squad functions
//*************************************************************************
```

```
//  blueDecide is the OODA loop for Blue
void BlueSquad::blueDecide (Squad * *blueActor, Squad * *redActor,
                                        const int blueTotal, const int
redTotal)
{
   if(agentState == ALIVE)
   {
      //  First step in OODA loop (Observe)
      surveyArea(blueActor, redActor, blueTotal, redTotal);
      riskArea(blueActor, redActor, blueTotal, redTotal);

      //  Second and Third step in OODA loop (Orient and Decide)
            if(timeStep%update == 0)
                  infoProcess();

      //  Fourth step in OODA loop (Act)
      shootDecision(blueActor, redActor, blueTotal, redTotal);
      move(fixx, fixy);
   }
}

   // surveyArea has Blue agent looking around him in a circle of
   //   size viewDistance.
void BlueSquad::surveyArea(Squad * *blueActor, Squad * *redActor,
                                    const int blueTotal, const int redTotal)
{
   redViewCount = 0;
   blueViewCount = 0;

   //  Loop scans for Red forces in area defined by ViewDistance, places all
   //    that meet criteria in a list
      for(int i = 0; i < redTotal; i++)
   {
      float distance = viewDistance;
      if((distance = (sqrt(pow((redActor[i]->position.x - position.x),2) +
            pow((redActor[i]->position.y - position.y),2)))) < viewDistance
            && redActor[i]->agentState == ALIVE)
      {
                  detection++;
            if (redViewCount < boundedRationality)
         {
            redList[redViewCount].id = redActor[i]->id;
            redList[redViewCount].positionx = redActor[i]->position.x+
                        randFunction(2*error) - error;
            redList[redViewCount].positiony = redActor[i]->position.y+
                        randFunction(2*error) - error;
            redList[redViewCount].direction = redActor[i]->direction;
            redList[redViewCount].distance = distance;
            redViewCount++;
         }
      }
      }


   //  Loop scans for Blue forces in area defined by ViewDistance, places all
```

```
   //    that meet criteria in a list
      for(int i = 0; i < blueTotal; i++)
   {
      float distance = viewDistance;
      if((distance = (sqrt(pow((blueActor[i]->position.x - position.x),2) +
            pow((blueActor[i]->position.y - position.y),2)))) < viewDistance
            && blueActor[i]->agentState == ALIVE)
      {
            if (blueViewCount < boundedRationality)
         {
            blueList[blueViewCount].id = blueActor[i]->id;
            blueList[blueViewCount].positionx = blueActor[i]->position.x+
                        randFunction(2*error) - error;
            blueList[blueViewCount].positiony = blueActor[i]->position.y+
                        randFunction(2*error) - error;
            blueList[blueViewCount].direction = blueActor[i]->direction;
            redList[redViewCount].distance = distance;
            blueViewCount++;
         }
      }
   }
}     //  end blueDecide

   // Risk area has Blue agent looking around him in a circle of
   //   size by riskDistance.  Risk area defines the range used in
   //   the affinity parameter.
void BlueSquad::riskArea(Squad * *blueActor, Squad * *redActor,
                                    const int blueTotal, const int redTotal)
{
   redRiskCount = 0;
   blueRiskCount = 0;

   //  Loop scans for Red forces in area defined by riskDistance, places all
   //    that meet criteria in a list
      for(int i = 0; i < redTotal; i++)
   {
      if(sqrt(pow((redActor[i]->position.x - position.x),2) +
            pow((redActor[i]->position.y - position.y),2)) < riskDistance
            && redActor[i]->agentState == ALIVE)
      {
                  detection++;
            if (redRiskCount < boundedRationality)
         {
            redRisk[redRiskCount].id = redActor[i]->id;
            redRisk[redRiskCount].positionx = redActor[i]->position.x+
                        randFunction(2*error) - error;
            redRisk[redRiskCount].positiony = redActor[i]->position.y+
                        randFunction(2*error) - error;
            redRisk[redRiskCount].direction = redActor[i]->direction;
            redRiskCount++;
         }
      }
      }

   //  Loop scans for Blue forces in area defined by riskDistance, places all
   //    that meet criteria in a list
      for(int i = 0; i < blueTotal; i++)
```

```
   {
      if(sqrt(pow((blueActor[i]->position.x - position.x),2) +
            pow((blueActor[i]->position.y - position.y),2)) < riskDistance
            && blueActor[i]->agentState == ALIVE)
      {
            if (blueRiskCount < boundedRationality)
         {
            blueRisk[blueRiskCount].id = blueActor[i]->id;
            blueRisk[blueRiskCount].positionx = blueActor[i]->position.x+
                        randFunction(2*error) - error;
            blueRisk[blueRiskCount].positiony = blueActor[i]->position.y+
                        randFunction(2*error) - error;
            blueRisk[blueRiskCount].direction = blueActor[i]->direction;
            blueRiskCount++;
         }
      }
   }
}     // end riskArea


   //       InfoProcess takes the data obtained from surveyArea and Risk Area
   //  and determines average position of opposing forces.  Based on force
   //  projections, it decides whether to run, attack, stay put, or continue
   //  towards the goal.
void BlueSquad::infoProcess()
{
   float averagex = 0;
   float averagey = 0;
   float sumx = 0;
   float sumy = 0;

   if(blueRiskCount < aggressive && blueViewCount > 0 && redViewCount > 0)
   {
      for(int i = 0; i < blueViewCount; i++)
      {
         sumx = sumx + blueList[i].positionx;
         sumy = sumy + blueList[i].positiony;
      }
      averagex = sumx/(float)blueViewCount;
      averagey = sumy/(float)blueViewCount;

      fixx = averagex;
      fixy = averagey;
   }
   else if(redViewCount == 0)
   {
      fixx = goalx + randFunction(10) - 5;
      fixy = goaly + randFunction(10) - 5;
   }
   else if(((float)redViewCount/blueViewCount) > charge &&
                              ((float)redViewCount/blueViewCount) < runaway)
   {
      fixx = position.x + randFunction(10) - 5;
      fixy = position.y + randFunction(10) - 5;
   }
   else
      {
```

```
      for(int i = 0; i < redViewCount; i++)
      {
         sumx = sumx + redList[i].positionx;
         sumy = sumy + redList[i].positiony;
      }
      averagex = sumx/(float)redViewCount;
      averagey = sumy/(float)redViewCount;

      if(((float)redViewCount/blueViewCount) <= charge)
         {
         fixx = averagex;
         fixy = averagey;
      }
         else
      {
            fixx = 2*position.x - averagex;
            fixy = 2*position.y - averagey;
      }
   }
}  // end infoProcess

      //  shootDecision determines if the criteria is met to shoot.
void BlueSquad::shootDecision(Squad * *blueActor, Squad * *redActor,
                                 const int blueTotal, const int redTotal)
{

      if(aggressive == NO)
   {
      if(shooter == NO && (targeted == YES || redRiskCount != 0))
            shooter = YES;
   }
   else
   {
      if(shooter == NO && (targeted == YES || redViewCount != 0))
            shooter = YES;
   }

      if(shooter == YES && redViewCount != 0 && pk != 0)
   {
      for(int i = 0; i < redViewCount; i++)
         {
         redActor[redList[i].id - 1]->targeted = YES;
         float result;
         result = pk*(1 - (pow(redList[i].distance,2)/pow(viewDistance,2)));
         if((result/redViewCount) > ((float)rand()/(float)RAND_MAX))
         {
            hitsGiven = hitsGiven + 1;
            redActor[redList[i].id - 1]->hitsTaken =
            redActor[redList[i].id - 1]->hitsTaken + 1;
         }
         if(redActor[redList[i].id - 1]->hitsTaken >=
                  redActor[redList[i].id - 1]->maxHits)
                  redActor[redList[i].id - 1]->agentState = DEAD;
      }
      }
   return;
}     // end shootDecision
```

```
//*****************************************************************************
//                                    General Squad Functions
//*****************************************************************************
//  Functions used by both Blue and Red agents.

//  Move function for forces with a given target coordinate

void Squad::move(float newPosx, float newPosy)
{
      if(position.x < 5)
      {
         position.x = 5 + speed;
         position.y = position.y + randFunction(2*speed) - speed;
      }
      else if(position.x > 700)
      {
         position.x = 700 - speed;
         position.y = position.y + randFunction(2*speed) - speed;
      }
      else if(position.y < 5)
      {
         position.y = 5 + speed;
         position.x = position.x + randFunction(2*speed) - speed;
      }
      else if(position.y > 550)
      {
         position.y = 550 - speed;
         position.x = position.x + randFunction(2*speed) - speed;
      }
      else
      {
      float step = speed; //stepSize;
      float divisor = newPosx - position.x;
      if(fabs(divisor)< .1)
            divisor = .1;

      if(fabs(newPosy - position.y) <= 1 && fabs(newPosx - position.x) <= 1)
            return;
      //   if statement moves Blue closer to goal
      float arcTan = fabs(atan((double)(newPosy - position.y)
                        / divisor));
         if ((newPosy - position.y) >= 0)
            {
            if ((newPosx - position.x) >= 0)
            {
                  position.x = position.x + step*cos(arcTan);
                  position.y = position.y + step*sin(arcTan);
                  direction = arcTan*180/pi;
            }
               else
            {
```

```
                position.x = position.x - step*cos(arcTan);
                position.y = position.y + step*sin(arcTan);
                direction = arcTan*180/pi + 90;
            }
        }
            else
        {
                if ((newPosx - position.x) >= 0)
            {
                position.x = position.x + step*cos(arcTan);
                position.y = position.y - step*sin(arcTan);
                direction = arcTan*180/pi + 270;
            }
            else
        {
                position.x = position.x - step*cos(arcTan);
                position.y = position.y - step*sin(arcTan);
                direction = arcTan*180/pi + 180;
                }
            }
    }
}     // end move

      //  Places the initial position of forces for both Blue and Red
void Squad::initialPosition(int centerx, int centery, int sizex, int sizey)
{
   boxsizex = sizex;
   boxsizey = sizey;
   boxcenterx = centerx;
   boxcentery = centery;

   if((boxsizex == 0) || (boxsizey == 0))
   {
      cerr << "boxsizex or boxsizey = 0" << endl;
      return;
   }

      position.x = boxcenterx + (float)randFunction(boxsizex) -
                        (float)boxsizex/2;
      position.y = boxcentery + (float)randFunction(boxsizey) -
                        (float)boxsizey/2;

      fixx = position.x;
      fixy = position.y;
}


//  General purpose random function generator

int Squad::randFunction(int input)
{
   return rand()%input;
}
```

4. Sample Batch Mode Output

Blue Avg. X	Blue Avg. Y	Red Avg. X	Red Avg. Y	Blue Dead	Red Dead	Var Blue Y	Var Red Y	Blue Dead Var	Red Dead Var
274.719	102.446	300.536	298.636	0	0	1.933	1.697	0	0
272.6	158.7	301.8	298.6	0	0	2.105	1.412	0	0
276.9	215.7	301.7	292.8	0	0	2.22	2.62	0	0
288.9	266.6	301	291.2	0	0.1	2.953	4.603	0	0.3
295.1	293.9	301.9	298.1	0	0.1	8.975	6.205	0	0.3
290.8	300.6	297.7	300.2	0.1	0.1	13.51	7.873	0.3	0.3
276.2	306	292	301.6	0.1	0.1	10.45	8.301	0.3	0.3
256.6	308.1	285.5	303.3	0.4	0.2	14.11	8.925	0.6633	0.4
242.4	310	280.2	304.7	0.4	0.6	18.09	10.56	0.6633	0.6633
234.1	318.6	278.9	305.6	0.5	1.3	21.96	11.97	0.922	1.269
225.8	322	277	305.7	0.5	2.3	32.62	12.87	0.922	1.552
222.2	319.9	275.4	305.5	0.9	3	37.95	14.08	1.221	2
217.2	312.8	274.4	305	1	3.9	34.5	14.28	1.265	2.385
216.2	309.4	273.2	302.5	1.1	5.3	28.77	14.16	1.513	2.492
218.7	302.6	273.2	300.5	1.4	6.8	39.1	14.34	1.428	3.092
218.8	296	273.8	298.6	1.4	8.2	55.45	14.5	1.428	3.4
217.2	296.2	275.6	296.6	1.5	9.2	65.81	14.33	1.36	3.919
218	302.9	276.9	295.2	1.8	10.8	67.11	14.26	1.939	4.556
226.8	319.4	278.1	294.4	1.8	12.3	67.39	14.4	1.939	3.796
235.4	337.8	278.8	294	2.1	13.3	70.05	14.3	2.468	3.689
245.2	354.2	279.7	293.9	2.1	14.4	75.33	14.43	2.468	3.72
259.3	374	280	294.9	2.1	14.7	73.48	16.02	2.468	3.348
275.8	389.7	280	295.6	2.2	15.4	73.95	16.6	2.482	2.939
290	402.4	279.9	296.1	2.4	16.1	73.79	17.41	3.04	2.809
299	410.5	279.6	297	3.1	17	67.67	18.48	3.754	3 13
305	419.4	280.3	297.7	3.4	17.4	62.66	18.67	3.8	3.007
306.3	426.9	279.7	297.7	3.4	17.5	60.93	18.75	3.8	3.074
307.5	430.7	278.8	297.5	3.5	17.7	58.39	18.72	4.08	3.466
309.8	434.1	278.9	297.4	3.5	18.1	55.78	18.71	4.08	4 061
312.2	437.3	279.1	297.2	3.6	18.1	54.91	18.69	4.03	4.061
312.5	440.2	278.7	297	3.6	18.3	54.62	18.78	4.03	4.383
312	442.9	278.1	296.5	3.6	18.5	54.22	18.83	4.03	4.365
309.3	442.1	278	296.5	3.7	18.9	55.61	18.91	4.001	4 742
307	444.6	278	296.5	3.7	19.1	54.33	19.12	4.001	5 108
304.5	448	278	296.5	3.7	19.2	50.76	19.31	4.001	5.307
303.3	451.1	277.8	296.3	3.7	19.2	48.88	19.26	4.001	5.307
302.5	453.4	278	296.1	3.7	19.2	48.99	19.1	4.001	5.307
301.6	455.9	278.3	295.6	3.7	19.2	50.6	18.68	4.001	5.307

LIST OF REFERENCES

[AXL97] Axelrod, Bob, *The Complexity of Cooperation : Agent-Based Models of Competition and Collaboration*, Princeton Studies in Complexity, 1997.

[BAR97] Bar-Yam Yanneer, Dyanmcis of Complex Systems, Addison-Wesley, 1997.

[BRA98] Brandstein, A. G. and Horne, G. E., *Data Farming: a Meta-technique for Research in the 21st Century*, Maneuver Warfare Science, Draft, 1998.

[CAS97] *Would-be Worlds: How Simulation is Changing the Frontiers of Science*, Casti, John L., New York, N. Y., J. Wiley & Sons Inc., 1997.

[CEB95] *Network-Centric Warfare: Its Origin and Future*, A. K. Cebrowski, J. J. Gartska, Proceedings, January 1998.

[HOL95] Holland, John H., *Hidden Order: How Adaptation Builds Complexity*, Addison-Wesley Publishing Company, Inc., 1995.

[HOR97] Horne, Gary E. and Leonardi, M. L., *Trust on the Battlefield: Exploring Questions with a New Tool*, Maneuver Warfare Science, Draft, 1998.

[ISA97]*Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial Life Approach to Land Warfare*, Center for Naval Analysis, 1997.

[JCS94] *Joint Pub 1-02: Department of Defense Dictionary of Military and Associated Terms*, Washington, D.C., Government Printing Office, March 23, 1994.

[KLE89] Klein, Gary A., *Strategies of Decision Making*, Military Review, 1989.

[MOR95] Braken, J., Kress, M., Rosenthal, R. E., editors, *Warfare Modeling*, John Wiley and Sons, Inc., 1995.

[MCD96] *Comand and Control, Marine Corps Doctrinal Publication*, PCN 142 000001 00, 4 October 1996.

[SFI95] Cowan G. A., Pines, D., Meltzer, D., editors, *Complexity: Metaphors, Models, and Reality*, Addison-Wesley, 1994.

[STO98] *Ship-to-Objective Maneuver Concept*, Surface Warfare, January/February 1998.

[WAL92]Waldrop, M. Mitchell, *Complexity: The Emerging Science at the Edge of Order and Chaos*, New York, NY, Simon and Schuster, 1992.

[ZIM98] Zimm, Alan D., *Modeling Maneuver Warfare: Incorporating Human Factors and Decision making in Combat Analysis*, John Hopkins University Applied Physics Laboratory, 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. VADM Arthur K. Cebrowski 1
   CNO (N6)
   2000 Navy Pentagon, Room 4E522
   Washington DC 20350-20000

4. Dr. Alfred Branstein 1
   Studies and Analysis Division
   3300 Russell Rd.
   Quantico, VA 22134-5130

5. Dr. Tom Czerwinski 1
   National Defense University
   Fort Lesley J. McNair
   Bldg 62, Room 212A
   Washington DC 20319

6. Dr. Andy Ilachinski 1
   Center for Naval Analysis
   PO Box 16268
   4401 Ford Avenue
   Alexandria, VA 22302-8268

7. Professor Donald P. Gaver, Jr. (Code OR/GV). 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

8. Professor Carl R. Jones (Code SM/JS) 1
   Naval Postgraduate School
   Monterey, CA 93943-5002

9. Capt. James R. Powell, USN (Code IW/PI) 1
   Naval Postgraduate School
   Monterey, Ca 93943-5002

10. Professor William Kemple (Code OR/KE) 1
    Naval Postgraduate School
    Monterey, CA 93943-5002

11. Professor Dan C. Boger (Code CC/BO) 1
    Naval Postgraduate School
    Monterey, CA 93943-5002

12. LT Richard B. Hencke 1
    USS Dubuque (LPD-8)
    FPO AP 96663-1711

15 483NPG TH 3436
10/99 22527-200

DUDLEY KNOX LIBRARY
3 2768 00365704 0